Bachelor-Thesis

XBRL basiertes Reporting as a private Service für Meldungen zur Bankenaufsicht

Vincent Mang

Ritterstr. 14

77652 Offenburg

Angewandte Informatik

Betreuer

Prof. Dr.-Ing. Daniel Fischer, Hochschule Offenburg

Dipl.-Inform. Björn Richter, EXXETA AG, Karlsruhe

Bearbeitungszeitraum

01.09.2015 bis 25.02.2016

Hochschule Offenburg

Fakultät: Elektrotechnik und Informationstechnik

Studiengang: Angewandte Informatik

Kurzfassung

Das Ziel der vorliegenden Bachelorarbeit ist es, ein prototypisches Meldesystem für die Bankenaufsicht in einer privaten Cloud zu realisieren. Ferner sollen die Vorteile eines solchen Systems gegenüber bestehenden Systemen abgegrenzt werden.

Im ersten Teil wird auf die Entwicklung von Software as a Service-Anwendungen sowie die rechtlichen Grundlagen und Zusammenhänge im Meldewesen eingegangen. Des Weiteren werden das Cloud-Computing und das verwendete Datenformat XBRL durchleuchtet. Anschließend werden bestehende Meldesysteme verglichen. Dabei stehen die unterschiedlichen Bereitstellungsarten im Fokus.

Zur Erreichung des Ziels der Thesis wird ein Proof of Concept anhand einer Beispielmeldung aus dem Großkreditmeldewesen umgesetzt. Diese Meldung erfordert sowohl das Befüllen von PDF-Formularen sowie das Erstellen von XBRL-Dateien. Für die Umsetzung wird eine Testumgebung aufgebaut, mit welcher es bedingt möglich ist, reale Bedingungen zu simulieren. Nach einer Evaluation wird hierfür das Platform as a Service-System Openshift verwendet. Damit ist es möglich, eine Private Cloud zu erstellen, auf der eine Java EE Anwendung als Reporting as a Service-Lösung zur Verfügung gestellt wird. Deren Funktionsumfang umfasst das Einlesen, Prüfen, Konvertieren und Validieren von Daten sowie das Versenden von Meldungen zur Bankenaufsicht. Um die heterogenen Systeme im Finanzumfeld abzudecken, werden Testdaten als CSV, XLS und via Datenbank bereitgestellt. Die Tests mit diesen Daten sind sehr positiv. So ist es möglich, Daten aus allen Datenquellen einzulesen und mittels einer Plausibilitätsliste der Bundesbank zu prüfen. Anschließend können die Daten in eines der geforderten Formate PDF oder XBRL konvertiert werden. Durch den Ansatz einer privaten Cloud-Lösung können einige Sicherheitsbedenken ausgeschlossen werden, ohne auf die Vorteile wie Lastverteilung, Multitenancy, Skalierbarkeit und hohe Kompatibilität verzichten zu müssen.

Abschließend werden in einer Schlussbetrachtung die Ergebnisse vorgestellt und ein Ausblick für die Zukunft gegeben.

Eidesstattliche Erklärung

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelor-Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Karlsruhe, den 25.02.2016 _____

Unterschrift

Urheberrechtliche Freigabe für die Hochschule

Diese Bachelor-Thesis ist urheberrechtlich geschützt, unbeschadet dessen wird folgenden Rechtsübertragungen zugestimmt:

- der Übertragung des Rechts zur Vervielfältigung der Bachelor-Thesis für Lehrzwecke an der Hochschule Offenburg (§ 16 UrhG),
- der Übertragung des Vortrags-, Aufführungs- und Vorführungsrechts für Lehrzwecke durch Professoren der Hochschule Offenburg (§ 19 UrhG),
- der Übertragung des Rechts auf Wiedergabe durch Bild- oder Tonträger an die Hochschule Offenburg (§21 UrhG).

Inhaltsverzeichnis

1 Einleitung

Durch die Globalisierung rückt die Welt näher zusammen. Gerade in der Geschäftswelt ist dies deutlich zu spüren. Der Austausch von Daten zwischen Unternehmen wird dadurch immer wichtiger. Des Weiteren spielt der Austausch zwischen Unternehmen und dem Staat eine immer größere Rolle. Gerade im Meldewesen ist dies eine zentrale Komponente. Im Finanzumfeld müssen hierbei Daten zwischen den Finanzunternehmen und der Bankenaufsicht ausgetauscht werden. Die Meldungen müssen dabei nicht nur an nationale, sondern auch an internationale Instanzen getätigt werden. Dabei führen gesetzliche Rahmenbedingungen und steigende Anforderungen zu einem stetigen Wachstum der Komplexität. Dies hat zur Folge, dass es immer schwerer wird, die oft zeitkritischen Meldungen rechtzeitig zu erstellen und zu versenden.[1]

Die Schwierigkeit besteht allerdings nicht ausschließlich in der Komplexität oder der Menge der Meldungen, sondern auch in der Vielfalt an unterschiedlichen Kommunikationspartnern. Eine solche Problematik kann durch klar definierte Standards gelöst werden. Mit Hilfe dieser ist es beispielsweise möglich, weltweit auf das Internet zuzugreifen oder zu telefonieren. Hierbei werden durch Standards die unterschiedlichen Anbieter und Technologien zusammengeführt und ermöglichen so einen reibungslosen Austausch von Informationen.[2] Eine Standardisierung von Meldungen bedeutet, dass sowohl die Struktur als auch deren Format zu definieren sind. Nur dadurch ist es möglich, einheitliche Schnittstellen zu entwickeln und eine Automatisierung zu ermöglichen. Das Format XBRL, welches für eXtensible Business Reporting Language steht, wurde genau für diesen Zweck entwickelt und hat sich bereits in einigen Bereichen etabliert. Dieser auf der eXtensible Markup Language (XML) basierende Standard soll den Datenaustausch und die Vergleichbarkeit der Daten vereinfachen. Ebenfalls ist es damit möglich, Validationen durchzuführen und Business Reports zu erstellen. Somit bietet es Vorteile für alle Bereiche, in denen das Sammeln, Analysieren, Managen oder Verbreiten von Business Informationen eine Rolle spielt.[3]
Diese Bereiche sind im Finanzsektor sehr wichtig, weshalb XBRL dort zunehmend an Bedeutung gewinnt.

[1] Vgl. Günther 2014, S. 124
[2] Vgl. Flickinger 2013, S. 17
[3] Vgl. Hoffman und Watson 2010, S. 25–26

1.1 Ausgangssituation

Die im Rahmen des Meldewesens erzeugten Meldungen werden zurzeit noch in unterschiedlichen Formaten an die Bundesbank gemeldet. Früher mussten alle Meldungen in Papierform erfolgen. Durch die Einführung elektronischer Eingabemasken wurden Meldungen per XML möglich. Aktuell sind sowohl Meldungen im Papier-, XML- und auch im XBRL-Format im Einsatz. In Zukunft sollen allerdings alle Meldungen schrittweise auf das XBRL-Format portiert werden.[4] Dies kommt daher, dass die internationalen Bankenaufsichtsinstanzen, wie die European Bank Authority (EBA), zunehmend mehr Meldungen von den nationalen Instanzen übernehmen.

Der Meldevorgang innerhalb der Finanzinstitute erfolgt aktuell sehr unterschiedlich. Dieser erstreckt sich vom manuellen Melden bis hin zu vollwertigen Meldesystemen. Oft bildet sich allerdings eine sogenannte Schatten-IT. Konkret bedeutet dies, dass mittels Excel oder Access eigene Lösungen oder Teillösungen erstellt werden. Diese sind allerdings oft sehr komplex, unübersichtlich und fehleranfällig.[5] Gerade bei kleinen oder mittelständischen Finanzunternehmen ist das häufig der Fall, da dort die Meldesysteme zu teuer sind. Auch das manuelle Melden kann sehr teuer werden. Dabei fallen zwar keine Lizenzkosten an, es wird allerdings viel Arbeitszeit gebunden oder es muss gegebenenfalls eine weitere Arbeitskraft beschäftigt werden.

Bei den aktuellen Meldesystemen werden oft unterschiedliche Schnittstellen zur Anbindung der Daten angeboten, um flexibel auf die Infrastruktur der Finanzinstitute eingehen zu können. Dies wird auch weiterhin nötig sein, da zwar die externe Schnittstelle standardisiert wurde, aber innerhalb der Banken weiterhin eine sehr heterogene Umgebung herrscht.

Die Datenhaltung erfolgt meist in Datenbanken. Daher bietet es sich an, eine Datenschnittstelle bereitzustellen. Falls dies nicht der Fall ist, bieten die meisten Datenhaltungssysteme eine Exportfunktion, mit der sich Daten in strukturierten Dateiformaten, wie z.B. Comma-separated values (CSV) oder einem Excel Spreadsheet (XLS/XLSX), exportieren lassen. Daher ist auch eine Schnittstelle für Dateien dieser Art sinnvoll.

Wie bereits beschrieben, müssen die Daten vor allem im XBRL-Format und in Papierform gemeldet werden. Daher bietet es sich an, diese beiden Ausgabeformate zu unterstützen.

[4] Vgl. Günther 2014, S. 266
[5] Vgl. Günther 2014, S. 123

1.2 Zielbestimmung

Das Ziel ist es, die Anforderungen im Meldewesen anhand eines Beispiels zu evaluieren. Dabei sollen sowohl regulatorische als auch sicherheitskritische Bedingungen berücksichtigt werden. Des Weiteren wird das Konzept für eine Reporting as a Service (RaaS) Lösung ausgearbeitet und prototypisch umgesetzt. Die Umsetzung als Software as a Service (SaaS) ist für diesen Anwendungsfall sehr vorteilhaft. Neben den offensichtlichen Vorteilen, wie Skalierbarkeit und der Nutzung „On-Demand“, wurden durch die Umstrukturierung des Meldewesens, wie in Abbildung 1 zu sehen, viele abteilungsübergreifenden Abhängigkeiten geschaffen. Diese bestehen vor allem zwischen Rechnungswesen, Controlling und dem Meldewesen selbst.[6] Da es somit nötig ist, mehreren Abteilungen bzw. Benutzern Zugriff zur gleichen Meldung zu gewährleisten, war eine Software as a Service Lösung sehr naheliegend.

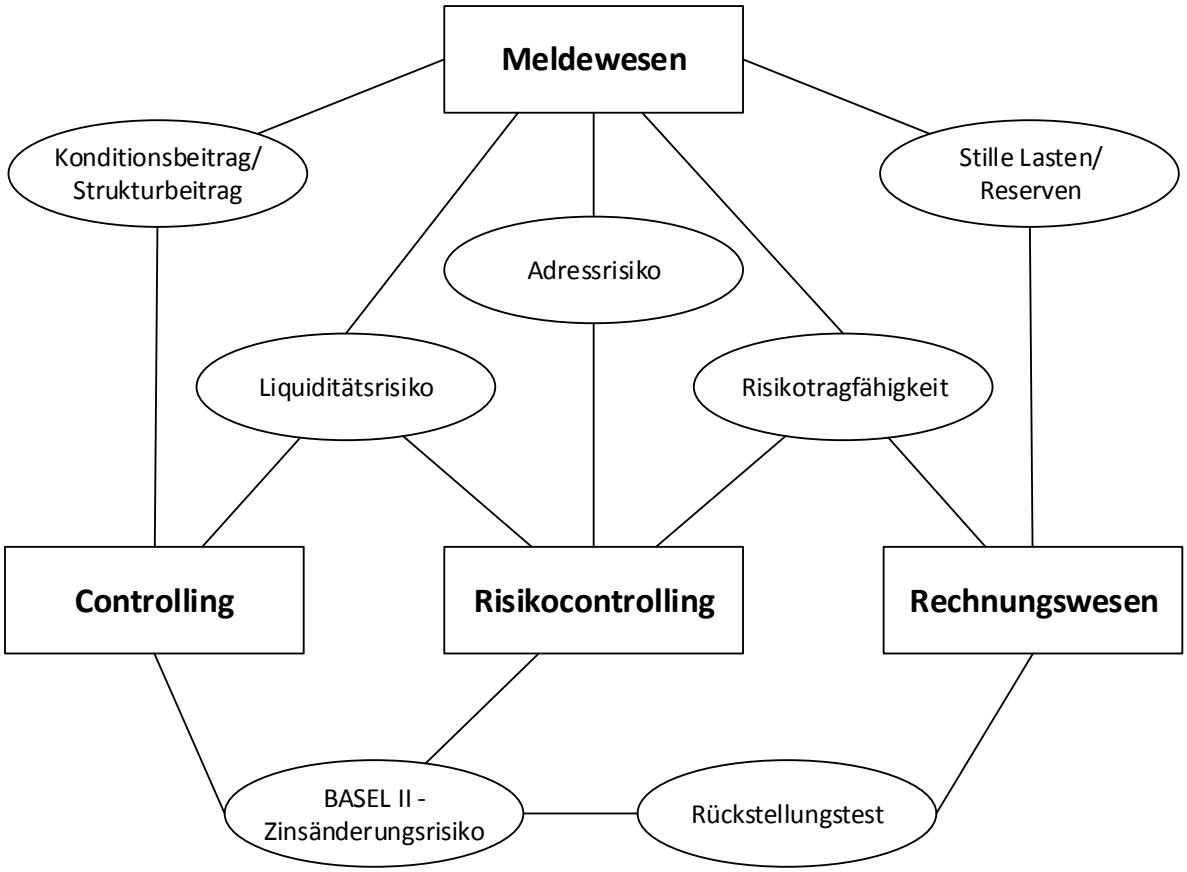

Abbildung 1: Zusammenhänge des Meldewesens[7]

Mit dieser Lösung soll es möglich sein, Daten aus unterschiedlichen Schnittstellen einzulesen und zu sammeln. Diese sollen dann entweder zu XBRL-Dateien konvertiert werden oder

[6] Vgl. Günther 2014, S. 115–116
[7] In Anlehnung an: Günther 2014

automatisch einen Meldevordruck befüllen. Da die Qualität und Validität der Daten eine sehr hohe Rolle spielt, ist das Validieren und Plausibilisieren dieser auch eine essenzielle Funktion der Anwendung. Als Ausgabeformate sollen XBRL-Berichte oder ausgefüllte Meldevordrucke geliefert werden. Durch diesen Proof of Concept (POC) sollen die Möglichkeiten des Smarten Reportings aufgezeigt und eine neue, flexible Lösung für das Meldewesen vorgestellt werden.

Die Bereitstellung dieser Cloud-Lösung soll vor allem durch Rechenzentren der Banken, wie z.B. der Finanz Informatik oder Fiducia & GAD, als private Cloud erfolgen. Aber auch der Einsatz in einer Community Cloud für FinTecs oder kleine Banken ist denkbar.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe und Technologien erklärt, welche in der vorliegenden Thesis von Bedeutung sind. Zu Beginn werden der Aufbau einer Cloud und die Abgrenzung der verschiedenen Cloudtypen erläutert. Danach wird der Begriff des Meldewesens im Reporting abgegrenzt. Abschließend wird das Datenformat XBRL detailliert betrachtet.

2.1 Cloud-Computing

Cloud-Computing ist das Resultat der technischen Evolution und wird bald als Selbstverständlichkeit angesehen werden, da es sich bereits jetzt unbemerkt in alltägliche Prozesse integriert. Beispiele hierfür sind Streaming-Dienste und Cloud-Speicher, die sehr verbreitet und beliebt sind.

Die eigentliche Entwicklung kann gut mit der Evolution der Wasserversorgung verglichen werden. Früher hatte jedes Dorf seine eigene Wasserversorgung und es war sehr mühsam, dieses Wasser zu beschaffen. Heutzutage ist es durch das geteilte, öffentliche Wassernetz nahezu überall möglich, sauberes Wasser, allein durch das Aufdrehen des Wasserhahns, zu erhalten. Wie beim Wasserversorgungsprozess können die Dienste beim Cloud-Computing schnell ein- und ausgeschaltet werden. Dabei steht ein Unternehmen im Hintergrund, welches dafür sorgt, dass die angebotenen Dienste sicher und jederzeit verfügbar sind. Auch die Ressourcen sind ähnlich wie beim Wasser verwaltet. Wenn der Hahn zu ist, wird nicht nur Wasser gespart, sondern man muss auch nichts dafür bezahlen. Dasselbe gilt auch für die Dienste der Cloud.[8]

Allgemein bezeichnet das Cloud-Computing das Speichern von Daten oder das Ausführen von Programmen auf entfernten Servern. Eine treffende Definition wurde von Baun et al. in „Cloud Computing“ geliefert:

> „Unter Ausnutzung virtualisierter Rechen- und Speicherressourcen und moderner Web-Technologien stellt Cloud-Computing skalierbare, netzwerkzentrierte, abstrahierte IT-Infrastrukturen, Plattformen und Anwendungen als on-demand Dienste zur Verfügung. Die Abrechnung dieser Dienste erfolgt nutzungsabhängig.“[9]

[8] Vgl. Kavis 2014
[9] Baun et al. 2011

Durch das National Institute of Standards and Technology (NIST) wurde ein Modell des Cloud-Computings erstellt, welches hohe Anerkennung genießt. Dieses Modell besteht aus fünf essenziellen Charakteristiken, vier Deployment Modellen und drei Service Modellen.[10] Diese werden in den folgenden Abschnitten erläutert. Eine Übersicht der Vor- und Nachteile des Cloud-Computing schließt dieses Unterkapitel ab.

2.1.1 Charakteristika

Eine Cloud sollte bestimmte Eigenschaften aufweisen, um sich als solche bezeichnen zu dürfen. Diese Eigenschaften werden als Charakteristika bezeichnet. Die folgenden Charakteristika werden als essenziell für eine Cloud betrachtet:

- Dienstbringung nach Bedarf: Nutzer können Dienste selbständig und ohne menschliche Interaktion seitens des Anbieters anfordern und nutzen. [11]
- Netzwerkbasierter Zugang: Dienste sind über Netzwerke mittels Standardtechnologien abrufbar. [12]
- Ressourcen-Pooling: Durch die Aufteilung der Ressourcen in Pools können diese parallel mehreren Nutzern zur Verfügung gestellt werden. Des Weiteren ist es dadurch möglich, die Ressourcen dem Bedarf der Nutzer anzupassen. [13]
- Elastizität: Ressourcen werden in feiner Granularität bereitgestellt. Dies ermöglicht eine hohe Skalierbarkeit und die nutzerseitige Illusion von unendlichen Ressourcen. [14]
- Messbare Dienstqualität: Dienste müssen quantitativ und qualitativ messbar sein. Dadurch ist es möglich, die Dienstqualität zu validieren, Transparenz zu schaffen und eine nutzungsabhängige Abrechnung zu gewährleisten.[15]

[10] Vgl. NIST Computer Security Division (CSD)
[11] Vgl. NIST Computer Security Division (CSD), Baun et al. 2011, Böttger 2012
[12] Vgl. NIST Computer Security Division (CSD), Baun et al. 2011, Böttger 2012
[13] Vgl. NIST Computer Security Division (CSD), Baun et al. 2011, Böttger 2012
[14] Vgl. NIST Computer Security Division (CSD), Baun et al. 2011, Böttger 2012
[15] Vgl. NIST Computer Security Division (CSD), Baun et al. 2011, Böttger 2012

2.1.2 Deployment Modelle

Ein Deployment Modell bezeichnet den Ort und die Verwendung der Cloud. Die vier gängigen Modelle werden in folgendem Abschnitt behandelt und unterstützend in Abbildung 2 visualisiert.

Public Cloud: Bezeichnet die bekannteste Cloudart. Diese zeichnet sich dadurch aus, dass sie öffentlich zugänglich ist und jeder die Services nutzen kann. Dabei werden im Regelfall Nutzungsgebühren erhoben. Die Verwaltung übernimmt in diesem Fall ein sogenannter Cloud Service Provider (CSP), also ein Unternehmen, welches sich darauf spezialisiert hat, Cloud-Dienste zu hosten. Dabei sind größere Unternehmen, wie z.B. Amazon oder Google meist attraktiver, da diese durch ihre Rechenzentrumsgröße eine sehr hohe Skalierbarkeit vorweisen und allgemein viele Ressourcen gewährleisten können.[16]

Private Cloud: Bei der privaten Cloud hingegen stehen die Dienste nur einem Unternehmen bzw. einer Organisation zur Verfügung. Bei der Verwaltung der Cloud gibt es hier jedoch unterschiedliche Modelle. Bei einer Insourced Private Cloud wird diese komplett im Unternehmen betrieben. Wird der Betrieb allerdings von einem externen Dienstleister übernommen, bezeichnet man dies als Managed Private Cloud. Wenn die komplette Verwaltung von einem Dienstleister übernommen wird, handelt es sich um eine Hosted Private Cloud. Dabei können Private Clouds sowohl On-Premise als auch Off-Premise sein.[17]

Community Cloud: Bei einer Community Cloud geht es vor allem um ein gemeinsames Ziel, welches eine Gruppe von Organisationen erreichen möchte. Des Weiteren teilen diese Organisationen gemeinsame Interessen, wie z.B. Sicherheitsmaßnahmen und regulatorische Vorschriften. Die Verwaltung kann hierbei entweder von den Organisationen selbst oder von einem externen Dienstleister übernommen werden. [18]

Hybrid Cloud: Ist eine Kombination der oben genannten Cloudarten. Dabei behalten die einzelnen Clouds ihre einzigartigen Identitäten, werden aber zu einer Einheit verbunden. Dabei ist es möglich, Daten und Programme zwischen den Clouds auszutauschen bzw. Zugriff darauf zu gewähren. Dies kann durch standardisierte oder proprietäre Technologien erfolgen. Anwendungsfälle sind hierbei das Cloud-Bursting und das Lastbalancing zwischen Clouds. [19]

[16] Vgl. Vossen et al. 2012, NIST Computer Security Division (CSD)
[17] Vgl. Barton 2014, Kavis 2014 Vossen et al. 2012
[18] Vgl. NIST Computer Security Division (CSD)
[19] Vgl. Barton 2014, Kavis 2014 Vossen et al. 2012Barton 2014

Die folgende Abbildung 2 visualisiert die Zusammenhänge und Einsatzgebiete der verschiedenen Deployment Modelle.

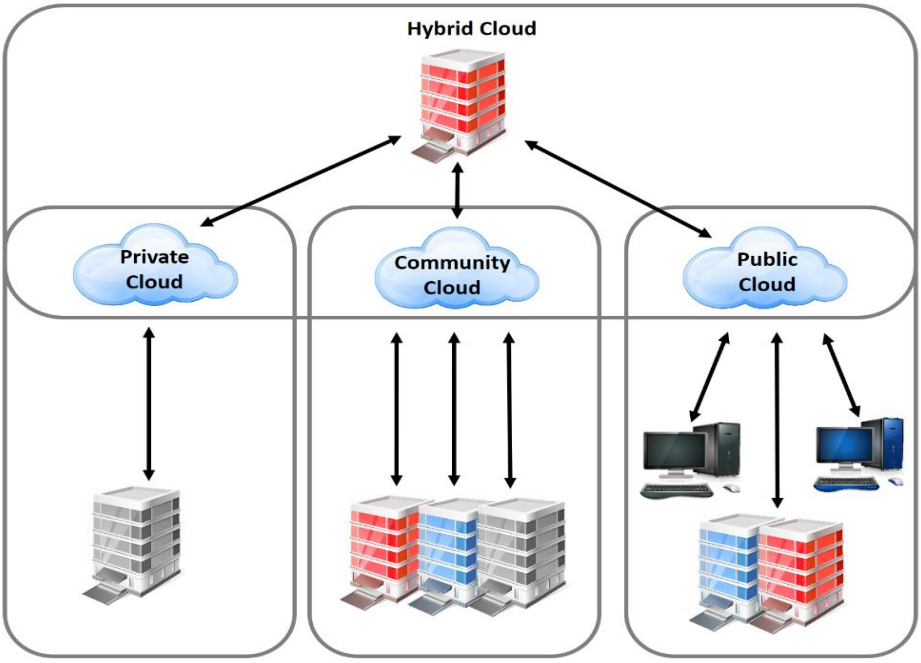

Abbildung 2: Visualisierung der Deployment Modelle[20]

Hier ist klar zu sehen, dass die Hybrid Cloud eine Kombination der anderen Deployment Modelle ermöglicht. Des Weiteren ist zu erkennen, dass die Private Cloud nur von einem Unternehmen genutzt wird. Die Community Cloud steht einem ausgewählten Benutzerkreis zur Verfügung. Die Public Cloud ist hingegen für alle verfügbar.

2.1.3 Service Modelle

Die Service Modelle beschreiben in Bezug auf die Abstraktionsebenen, welche Bereiche selbst und welche von der Cloud gemanagt werden müssen. Dabei wurden drei Service-Typen universell akzeptiert.[21]

Diese sind Infrastruktur as a Service (IaaS), Plattform as a Service (PaaS) und die bereits genannte Software as a Service, welche im Laufe des Kapitels erläutert werden. Die weitere Unterteilung der Servicekategorien ist in der Regel nicht trennscharf, da sich alle weiteren Unterteilungen auf PaaS, SaaS oder IaaS zurückführen lassen. Es sei lediglich noch die Kategorisierung Reporting as a Service genannt, welche aufgrund der Relevanz für diese Arbeit in einem eigenen Abschnitt behandelt wird.

[20] Eigene Darstellung
[21] Vgl. Sosinsky 2011

Begleitend zu der folgenden Beschreibung veranschaulicht Abbildung 3 die Service Modelle und die darin enthaltenen Komponenten. Wie zu sehen ist, mussten bei den traditionellen IT-Infrastrukturen alle Komponenten selbst gemanagt werden. Durch die verschiedenen Service Modelle ist es nun möglich, gewisse Teile auszulagern.

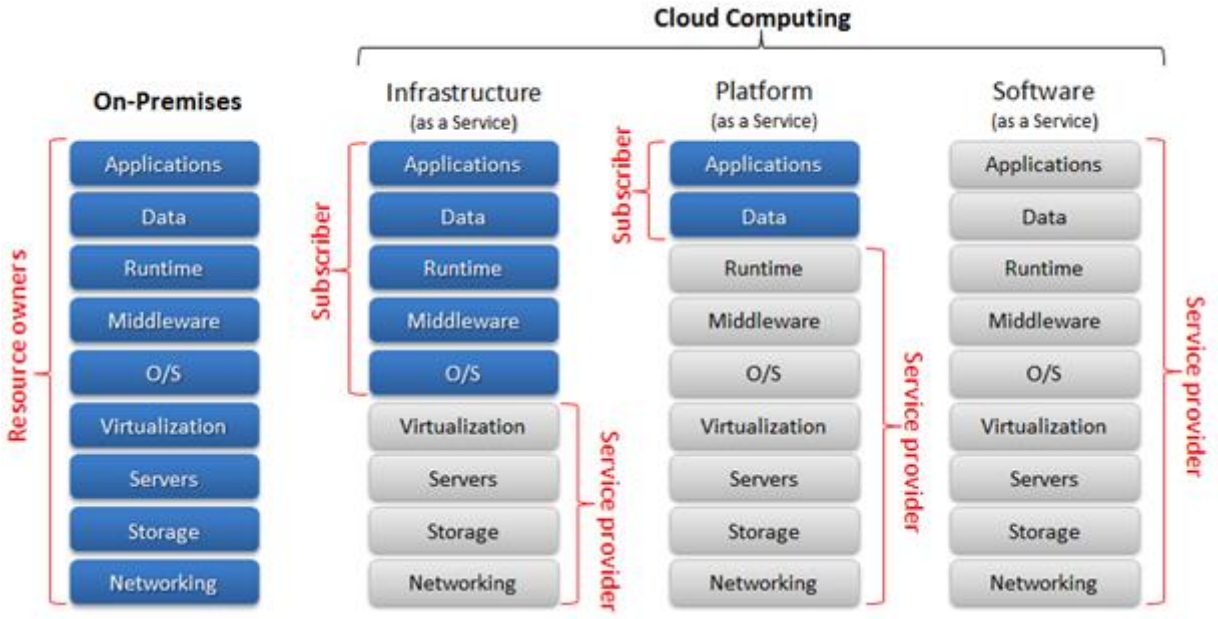

Abbildung 3: Abstraktionsebenen der Service Modelle [22]

Bei der **Infrastructure as a Service** bietet der Cloud-Provider virtuelle Hardware oder Infrastrukturdienste an, wie z.B. Speicherplatz, Rechenleistung oder Netzwerkbandbreite. Wie in Abbildung 3 zu sehen ist, gehört auch die Virtualisierung zu den Aufgaben des Providers. Dem Cloud-Nutzer wird also eine virtuelle Infrastruktur zur Verfügung gestellt. Diese hat neben dem Entfallen der Anschaffungskosten für die Hardware die Vorteile, dass sie hochverfügbar ist und die Backups und Wartungsarbeiten vom Anbieter übernommen werden. Die Benutzung der Ressourcen ist dabei mit größtmöglicher Flexibilität möglich. Dafür muss der Nutzer allerdings alle Schichten oberhalb der Infrastrukturschicht, wie z.B. Betriebssystem, Web- und Datenbankserver, selbst verwalten.[23]

Die nächste Ebene wird durch die **Plattform as a Service** abgedeckt. Dabei werden, wie in Abbildung 3 zu sehen, zusätzlich die Komponenten O/S, Middleware und Runtime vom Cloud-Provider übernommen. Der Nutzer muss sich also nur noch um die Daten und Anwendungen

[22] entnommen aus (cloudlighthouse.be, 2015)
[23] Vgl. Vossen et al. 2012

kümmern. Dies bietet die Möglichkeit, eigene Programme auf einer Plattform in der Cloud bereitzustellen und zu entwickeln.

Der Provider legt dabei gewisse Rahmenbedingungen fest, wie z.B. die Programmiersprache, Bibliotheken, Datenbanken, Applikationsserver oder Schnittstellen. Der Kunde kann innerhalb dieses Rahmens seine Programme frei gestalten. Oft wird die Plattform nicht nur für den Betrieb, sondern auch für die Entwicklung der Programme angeboten. Durch den festen Rahmen können gewisse Automatisierungsmechanismen ermöglicht werden. So ist es üblich, dass PaaS-Anbieter ein automatisches Loadbalancing anbieten. Außerdem können Kopien der Programme auf andere Systeme verteilt werden, sodass eine sehr hohe Verfügbarkeit und Skalierung gewährleistet werden kann.[24]

Als **Software as a Service** wird letztlich eine Anwendung bezeichnet, welche vom Nutzer direkt verwendet werden kann. Der Betrieb der Software liegt dabei vollständig beim Anbieter, der sich um die Wartung, Aktualisierung, Fehlerbeseitigung und Weiterentwicklung der Software kümmert. Durch den Einsatz von mandantenfähigen Programmen ist eine Skalierung möglich. Für den Zugriff auf die Software benötigt der Benutzer lediglich einen Webbrowser. Dort muss der Benutzer nur seine Daten ins System eintragen und Konfigurationen zur Individualisierung vornehmen. Backups der Daten und Software-Updates erfolgen in der Regel im Hintergrund und vom Benutzer unbemerkt.[25]

Reporting as a Service bezeichnet den Bereich der SaaS, welcher sich mit Berichten befasst. Im Rahmen dieser Thesis wird eine Anwendung in ebendiesem erstellt und vorgestellt.

Wie in der folgenden Abbildung 4 zu sehen ist, sind die Systeme sehr gut geeignet, um aufeinander aufzubauen. Dies ist allerdings nicht zwingend nötig. So muss zur Entwicklung einer SaaS nicht unbedingt eine PaaS verwendet werden. Um ein besseres Verständnis für die Ebenen zu entwickeln werden in Abbildung 4 bekannte Beispiele zu den verschiedenen Service Modellen gezeigt.

[24] Vgl. Vossen et al. 2012
[25] Vgl. Vossen et al. 2012

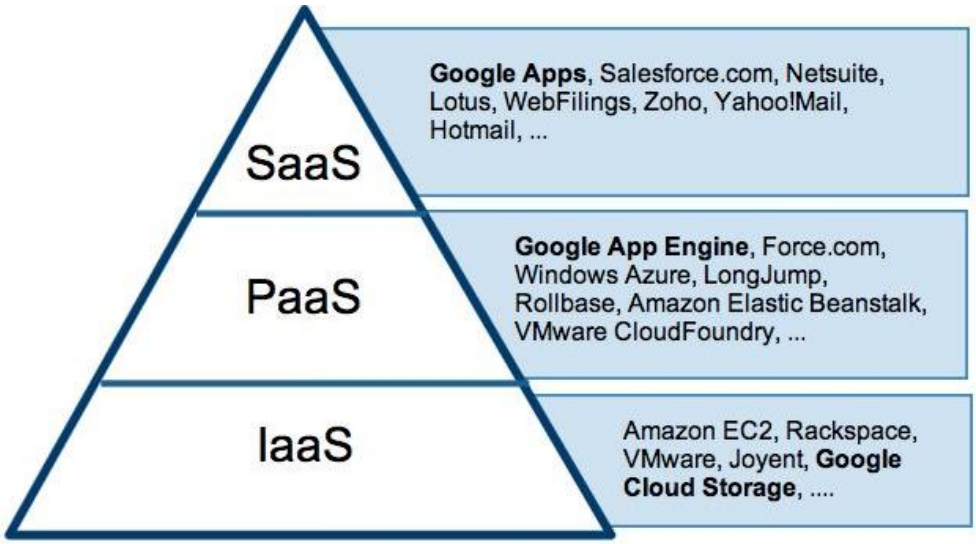

Abbildung 4: Aufbau Service Modelle[26]

Wie in Abbildung 4 erkennbar, kann ein Anbieter einer Anwendung auf die Lösung eines PaaS-Hosters anknüpfen, um die eigene Anwendung in der Cloud als SaaS Anbieter bereitzustellen, ohne selbst eine entsprechende Infrastruktur betreiben zu müssen. Eine strikte Einhaltung dieser Wertschöpfungskette ist jedoch nicht zwingend notwendig. Endkunden könnten beispielsweise auch direkt auf IaaS-Lösungen zurückgreifen.

2.1.4 Vor- und Nachteile

Die Vorteile des Cloud-Computings liegen in vielen Bereichen. Dieser Abschnitt bietet eine Übersicht über die Vor- und Nachteile, welche durch eine Cloud entstehen. Einige der Vor- bzw. Nachteile sind abhängig vom jeweiligen Deployment oder Service Modell. So kann es vorkommen, dass gewisse Eigenschaften bei den unterschiedlichen Modellen in ungleicher Intensität enthalten sind oder gar wegfallen. Da in dieser Thesis nur die Privaten und Community Clouds von Bedeutung sind, wird nur auf diese näher eingegangen. Dabei wird von vertrauenswürdigen CSPs und Rechenzentren, welche ausschließlich in Deutschland angesiedelt sind, ausgegangen.

Die genannten Einschränkungen sind aufgrund der Sicherheitsbestimmungen, welche in Kapitel 3.2 behandelt werden, nötig. Dadurch entfallen auch einige Nachteile, wie Sicherheitsprobleme und Datenschutzbestimmungen. Des Weiteren wird dadurch die Latenzproblematik teilweise entschärft.

[26] entnommen aus (/techinerd.com, 2015)

Einer der Vorteile des Cloud-Computings sind die Kostensenkungen. Diese ergeben sich nicht nur durch die Effizienzsteigerungen. Allein dadurch, dass die Abrechnung der Ressourcen im Wesentlichen anhand der tatsächlichen Nutzung erfolgt, entfallen Ausgaben für Sicherheitsreserven und ungenutzte Kapazitäten, die im eigenen Rechenzentrum vorgehalten werden müssten.
Des Weiteren sind die Personalkosten durch das Entfallen von Aufgaben wie Backups, Wartung, Inventarisierung usw. wesentlich niedriger oder entfallen sogar vollständig. Neben Personalkosten lassen sich zudem Einsparungen bei der Beschaffung von Hardware und Software realisieren. Hardware und Software verursachen nur Kosten, wenn diese tatsächlich genutzt werden. Dadurch entstehen keine langfristigen Kapitalbindungen und die Kostenplanung wird vereinfacht.
Die Skalierbarkeit der Dienste ist ebenfalls ein sehr wichtiger Vorteil. Dabei können Ressourcen bedarfsweise angemietet werden. Damit können Nutzungsspitzen ausgeglichen oder schnell auf Wachstum reagiert werden. Dies vereinfacht die Kapazitätsplanung stark. [27]

Prinzipiell bringt eine Cloud mehr Vorteile für kleine und mittelständische Unternehmen, da diese nicht die Ressourcen für eine eigene große IT-Infrastruktur oder speziell entwickelte Individualsoftware aufbringen können. Weitere Nachteile werden in den folgenden Absätzen erläutert.
Der Cloud-Service im Allgemeinen ist in einigen Fällen nicht so anpassbar oder individualisierbar, wie es gewünscht wird. Auch kann es vorkommen, dass die Cloud Anwendungen nicht den gleichen Funktionsumfang bieten wie eine On-Premise Lösung.

Wie bei allen ausgelagerten Systemen kann es auch beim Cloud-Computing zu Latenzproblemen kommen. Dies ist gerade dann ein Problem, wenn große Datenmengen übertragen werden und eine Echtzeitantwort erwartet wird. Bei Private und Community Clouds ist allerdings meist für eine gute Anbindung gesorgt. Zudem sind Echtzeitantworten beim Reporting eher von geringer Priorität.[28]

Nachdem nun das Cloud-Computing und somit die technische Grundlage in den verschiedenen Ausprägungen erläutert und die Vor- und Nachteile verdeutlicht wurden, werden im nächsten Kapitel die Grundlagen des fachlichen Teils erläutert.

[27] Vgl. Vossen et al. 2012
[28] Vgl. Sosinsky 2011

2.2 Meldewesen als Teilbereich des externen Rechnungswesens

Im Finanzbereich wird der Bereich als Meldewesen bezeichnet, welcher für die Erfüllung gesetzlicher Meldepflichten zuständig ist. Das Zusammenspiel der verschiedenen Bereiche des Meldewesens und der beteiligten Instanzen wird in diesem Abschnitt erläutert.

Gerade in der Finanzwelt müssen viele Meldungen an nationale Kontrollinstanzen, wie z.B. die Deutsche Bundesbank oder die Bundesanstalt für Finanzdienstleistungsaufsicht, abgegeben werden. Aber auch internationale Aufsichtsinstanzen wie die European Bank Authority sind vorhanden. Diese europäische Aufsichtsbehörde hat seit Dezember 2011 mehrere Implementing Technical Standards (ITS) mitunter zum Common Reporting Framework (COREP) verfasst. Diese Standards sind seit dem 01.01.2014 in Europa verbindlich.[29]

Da die Meldungen unter technischen Gesichtspunkten sehr ähnlich sind, wurde ein konkretes Beispiel gewählt, um den Vorgang und die Umsetzung zu veranschaulichen. Aufgrund ihrer simplen Auslegung war es anfänglich vorgesehen, die Meldungen zu den Millionenkrediten als Beispiel zu nehmen. Diese Meldung war zusammen mit den Großkrediten unter GroMiKV bekannt. Seit 2014 werden diese Meldungen allerdings separat und mit unterschiedlichen aufsichtsrechtlichen Grundlagen behandelt. Dies kommt daher, dass das Millionenkreditwesen weiterhin national mittels eines stufenweisen Modernisierungskonzeptes behandelt wird. Die Großkredite hingegen gehören bereits dem europäisch geregelten Solvenzmeldewesen und damit COREP an. Dadurch unterscheiden sich auch die Prozesse für die Erstellung der unterschiedlichen Meldungen.[30]

Da die Millionenkredite bisher nur national gemeldet werden, wird hier noch XML als Datenformat verwendet. Die Europäische Zentralbank ist allerdings bemüht, das Millionenkreditmeldewesen zukünftig zu internationalisieren. Dies soll mit Umsetzung der Analytical Credit Datasets stufenweise realisiert werden. Zeitlich soll das Vorhaben 2017 starten. Ziel der Internationalisierung ist es, die Datenqualität und Vergleichbarkeit der Meldungen zu erhöhen. Dies soll mittels der Umstellung auf XBRL erreicht werden. Details dazu hat die EU Kommission als Durchführungsverordnung Nr. 680/2014 veröffentlicht.[31] Genaueres zu den gesetzlichen Rahmenbedingungen wird in Kapitel 3.3 erläutert.

[29] Vgl. Günther 2014, S. 262
[30] Vgl. Günther 2014, S. 266
[31] Vgl. Günther 2014, S. 266

Parallel zur Inkraftsetzung der ITS werden im Millionenkreditwesen die Einreichungsformate verbindlich für Betrags- und Stammdaten auf XBRL abgeändert.

Damit soll es möglich werden, eine einheitliche Datenbasis für den internationalen Vergleich zu schaffen. Zudem wird dadurch die Möglichkeit für ein integriertes, intern und extern miteinander vernetztes, qualitativ hochwertiges Reporting geschaffen.[32]

Da sich dieser Vorgang noch einige Jahre hinzieht, wird der POC mittels der Meldungen zu den Großkrediten durchgeführt. Diese ist, wie bereits erwähnt, in COREP integriert und wird nach den neuen internationalen Standards gemeldet. Zusätzlich ist die Meldung zu den Großkrediten modular aufgebaut. Dadurch ist es möglich, nur einen Teil der Meldung für den POC zu verwenden, was zu einer wesentlichen Vereinfachung des fachlichen Teils führt. Ziel ist es allerdings, den POC so zu gestalten und zu beschreiben, dass auch andere Meldungen darauf abgebildet werden können.

2.3 XBRL als Standard für das Business Reporting

Wie bereits beschrieben ist XBRL ein globaler Standard für das Business Reporting.[33] Damit ist es möglich, die Lücke zwischen unterschiedlichen Systemen zu schließen und dadurch eine Effizienzsteigerung zu erreichen. Da dieser Standard auf XML basiert, ist das automatisierte Verarbeiten von Daten sehr einfach.[34]

XBRL besteht im Gegensatz zu XML nicht nur aus reiner Datentechnik, sondern auch aus dem Fachwissen aus den Bereichen Betriebswirtschaft und Rechnungswesen, welches stark mit der Datenstruktur verknüpft ist.[35] Die verschiedenen Schichten von XBRL sind in Abbildung 5 dargestellt und werden im Folgenden erläutert.

[32] Vgl. Günther 2014, S. 277
[33] Vgl. Hoffman und Watson 2010
[34] Vgl. Hoffman und Watson 2010
[35] Vgl. Flickinger 2013, S. 24

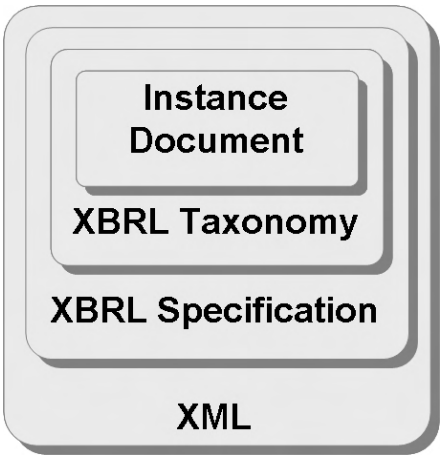

Abbildung 5: Schichten der XBRL [36]

Die äußerste Schicht ist XML. Diese bildet zusammen mit der XBRL Spezifikationsschicht die Grundlage, also die datentechnische Spezifikation des Dateiformates. Danach folgt die Taxonomie, in welcher das Fachwissen und die Datentechnik verknüpft werden. Die oberste Schicht ist das Instanz-Dokument, welches auch als XBRL-Report bezeichnet wird. Dies stellt die eigentliche Nutzlast dar, da dieses Dokument am Ende für den Datentransfer verwendet wird.[37] In folgendem Abschnitt wird auf das Instanz-Dokument und die Taxonomie, soweit es für das Verständnis der Thesis und der gewählten Meldung wichtig ist eingegangen.

Die wichtigsten Elemente, die ein Instanz-Dokument im Rahmen dieser Thesis enthält, sind *Tags*, *Namespaces*, ein *Header*, *Filing-Indicators*, *Kontext-Elemente* und *Dimensionen*. Dabei dienen die *Tags* ähnlich wie bei HTML zur Identifizierung des Elements. Die *Namespaces* werden als Präfix eingesetzt, um einen *Tag* eindeutig zuzuordnen. Dabei kann ein bereits definiertes Präfix mit Doppelpunkt getrennt vor den *Tag* geschrieben werden. Dies wird in Listing 1 dargestellt.

```
<präfix:TagID>Informations</präfix:TagID>
```

Listing 1: Beispiel eines XBRL-Tags mit Namespace

Der *Header* enthält Metainformationen wie den Namen des Sachbearbeiters und dessen Email-Adresse. Die darauffolgenden *Filing-Indicators* geben an, welche Meldungen in dem entsprechenden Dokument enthalten sind. Anschließend folgen die eigentlichen Meldungen, die sich aus speziellen *Tags* zusammensetzen. Dabei enthält jede Meldung ihr eigenes *Kontext-Element*, welches angibt, in welchen Kontext diese Meldung einzuordnen ist.

[36] Vgl. Flickinger 2013, S. 24
[37] Vgl. Flickinger 2013, S. 24–25

Dieses *Kontext-Element* enthält eine Identifikationsnummer, die zeitlichen Rahmenbedingungen und die Dimensionen. Mit den *Dimensionen* kann die Meldung in gewisse Kategorien eingeteilt werden, die später vor allem zu Auswertungszwecken sehr wichtig sind.

Die Taxonomie ist ein Begriff aus der Informationstechnologie und wird verwendet, um Elemente zu klassifizieren und deren Zusammenhänge aufzuzeigen. Im XBRL-Bereich werden dort die Elemente aufgelistet, welche in einem Bericht vorkommen können. Außerdem können damit deren Eigenschaften und Beziehungen untereinander definiert werden.[38]

Sie besteht im Wesentlichen aus *Concepts* und *Linkbases*. Die *Concepts* beinhalten die Elemente, welche das Instanz-Dokument enthalten kann und deren Eigenschaften. Die *Linkbases* beinhalten Strukturinformationen und Zusammenhänge für die verschiedenen Elemente, welche optischer, ordnungstechnischer oder inhaltlicher Natur sein können. Damit ist es möglich, Reihenfolgen festzulegen, Elemente für einen Report anzuordnen oder rechnerische Zusammenhänge zu definieren.[39]

Der weitere Aufbau und die Zusammenhänge der verschiedenen Taxonomie-Elemente würden an dieser Stelle zu weit führen. Relevant ist allerdings, dass es mittels der Taxonomie möglich ist, das Instanz-Dokument zu prüfen und zu validieren. Dadurch ist es möglich, die Dokumente vor dem Versenden auf Richtigkeit der Struktur sowie Plausibilität der Daten zu prüfen.[40]

Nachdem nun alle Grundlagen erklärt wurden, werden im nächsten Kapitel die Probleme analysiert, welche während der Umsetzung zu beachten sind.

[38] Vgl. Flickinger 2013, S. 29
[39] Vgl. Flickinger 2013, S. 30
[40] Vgl. Lan 2014

3 Problemanalyse

In diesem Kapitel werden die Probleme analysiert, welche während der Konzeption und Umsetzung vorliegender Thesis beachtet werden müssen. Dies umfasst regulatorische, gesetzliche und technische Einschränkungen und Vorschriften, auf welche während der Bearbeitung geachtet werden muss.

3.1 Automatisiertes Melden von Großkrediten

Durch die ständig wachsende Zahl an Meldungen und die Steigerung der Komplexität sowie des Umfangs dieser, wird es für Banken immer schwerer, den Anforderungen gerecht zu werden.[41] Gerade für kleine und mittelständische Banken ist dies eine große Herausforderung. Aus der finanziellen Perspektive wird das Meldewesen immer mehr zum Kostentreiber. Dies erhöht zusätzlich die Dringlichkeit, den Meldeprozess zu optimieren. Zur Realisierung wird, wie in Kapitel 2.2 erwähnt, die Meldung der Großkredite an die Bundesbank gewählt, da diese den Meldevorgang sehr gut veranschaulicht, ohne einen zu großen fachlichen Overhead. Das hier beschriebene Vorgehensmodell kann auf andere Meldungen übertragen werden.

Durch das automatisierte Melden von Großkrediten wird der Meldevorgang effizienter gestaltet. Hierbei ist nicht nur eine Zeitersparnis möglich, sondern es ist auch eine geringere Fehlerwahrscheinlichkeit als beim manuellen Melden vorhanden.[42]

Das Melden von Daten an die Deutsche Bundesbank im Kontext von Großkrediten wird durch das Merkblatt für die Angabe der Groß- und Millionenkredite beschrieben. Im folgenden Abschnitt werden die relevanten Informationen daraus erläutert.

Grundsätzlich ist bei der Einreichung zwischen Stammdaten und Betragsdaten zu unterscheiden. Die Einreichung von Stammdaten ist nur in Papierform zulässig. Dabei ist darauf zu achten, dass die Formulare, welche von der Bundesbank zur Verfügung gestellt werden, zu verwenden sind. Betragsdaten hingegen können nur in elektronischer Form eingereicht werden.[43]

Die papierlose Einreichung der Betragsdaten ist prinzipiell über zwei Verfahren möglich. Zum einen über die Erfassungsplattform der Groß- und Millionenkredite und zum anderen über den Datei-Upload über das ExtraNet der Deutschen Bank.[44] Wirklich automatisieren lässt sich nur

[41] Vgl. Günther 2014
[42] Vgl. Günther 2014
[43] Vgl. Bundesbank 2009
[44] Vgl. Bundesbank 2009

Variante zwei. Somit ist für die vorliegende Arbeit nur diese von Bedeutung.
Um den angesprochenen Dienst nutzen zu können, müssen einige formale Anforderungen erfüllt werden. Zuerst muss eine Benutzerregistrierung beim ExtraNet durchgeführt werden. Des Weiteren ist für die Einreichung von Daten eine Registrierung der Fachverfahrensfunktion „Einreichung von Groß- und Millionenkreditanzeigen“ erforderlich. Als weitere Anforderung muss eine Einreichungserklärung abgegeben werden. Mit dieser wird auf die rechtsverbindliche Unterschrift verzichtet und die beleglose Einreichung verbindlich anerkannt.[45]
Wenn diese Anforderungen erfüllt sind, können die Betragsdaten, im bereits beschriebenen XBRL-Format, elektronisch über das ExtraNet gemeldet werden.

Da die Stammdaten nur in Papierform eingereicht werden dürfen, ist hier nur eine Teilautomatisierung möglich. Hierbei können die bereitgestellten Formulare automatisch mit Daten befüllt und ausgedruckt werden. Das Versenden muss weiterhin manuell erfolgen.

Der gesamte Meldeprozess gestaltet sich in folgender Weise:

- Die Stammdatenanzeige in Papierform muss kontinuierlich bis zum 15. Geschäftstag nach dem Meldestichtag erfolgen.
- Nach der Verarbeitung der Stammdaten sendet die Bundesbank eine Stammdatenrückmeldung an die Banken am 25. Geschäftstag im XBRL-Format.
- Einreichung der Betragsdaten auf Basis der ITS-Vorgaben bis zum 30. Geschäftstag per File-Transfer.[46]

Die Einreichung der Stammdaten in elektronischer Form war von der Bundesbank für Anfang 2016 geplant. Dies wurde allerdings auf unbestimmte Zeit verschoben. Sobald die Einreichung umgestellt ist, wird sich der Meldeprozess voraussichtlich ändern, da die Verarbeitung und Lieferung der Stammdatenmeldung wesentlich weniger Zeit in Anspruch nehmen wird.

Nachdem die Rahmenbedingungen an die Meldung erläutert wurden, sind im nächsten Unterkapitel die Bedingungen an das technische Umfeld beschrieben.

[45] Vgl. Bundesbank 2009
[46] Vgl. Günther 2014, S. 29

3.2 Anforderungen an die IT-Infrastruktur im Bankenumfeld

Im Bankenumfeld müssen spezielle Vorschriften bezüglich der Sicherheit im IT-Umfeld berücksichtigt werden. Die Vorschriften werden in diesem Abschnitt analysiert. Dabei beschränkt sich die Analyse auf Private oder Community Clouds, welche z.B. bei den Rechenzentren der Banken, wie Finanz Informatik oder Fiducia & GAD, angesiedelt sind, da diese dem Einsatzbereich der in dieser Thesis entwickelten Anwendung entsprechen.

Die Anforderungen der Banken betreffend der IT-Infrastruktur sind grundsätzlich dem §25a des Kreditwesengesetzes (KWG) zu entnehmen. Die dort relevanten Anforderungen werden in den Mindestanforderungen an das Risikomanagement (MaRisk) genauer definiert.[47] Da das Cloud Computing an sich dort noch nicht aufgeführt ist, kann vergleichsweise das Outsourcing, welches im Allgemeinen Teil (AT) 9 der MaRisk beschrieben wird, herangezogen werden.[48]

Dabei muss vor allem sichergestellt sein, dass die datenschutzrechtlichen Bestimmungen eingehalten werden. Außerdem gilt es, Vorkehrungen zu treffen, welche, bei Beendigung der Auslagerung, das Fortbestehen der ausgelagerten Aktivitäten und Prozesse gewährleisten.[49]

Des Weiteren muss der AT 7.2 berücksichtigt werden, welche unter anderem besagt:

> „IT-Prozesse müssen die Integrität, die Verfügbarkeit, die Authentizität sowie die Vertraulichkeit der Daten sicherstellen. Für diese Zwecke ist bei der Ausgestaltung der IT-Systeme und der zugehörigen IT-Prozesse grundsätzlich auf gängige Standards abzustellen, insbesondere sind Prozesse für eine angemessene IT-Berechtigungsvergabe einzurichten, die sicherstellen, dass jeder Mitarbeiter nur über die Rechte verfügt, die er für seine Tätigkeit benötigt“[50]

Die darin genannten „gängigen Standards“ werden im Allgemeinen vom Bundesamt für Sicherheit in der Informationstechnik (BSI) definiert. Dabei werden unter anderem Angaben zur Cloud-Nutzung und zum Cloud-Management gemacht. Diese BSI Empfehlungen werden im folgenden Abschnitt aufgegriffen.

[47] Vgl. BaFin 2012
[48] Vgl. Niemann et al. 2014
[49] Vgl. BaFin 2012
[50] BaFin 2012

Bei der Community Cloud ist nur die Cloud-Nutzung zu betrachten, wobei vor allem das Definieren von Service-Level-Agreements (SLA) im Fokus steht. Damit können klare Abgrenzungen und Regeln zwischen CSP und Kunden festgelegt werden. Zudem werden regelmäßige Audits des Providers empfohlen. [51]

Bei einer Private Cloud muss zusätzlich der Cloud-Management Teil betrachtet werden.[52] In diesem werden Empfehlungen zur Absicherung der Cloud-Umgebung und der unterschiedlichen Instanzen gegeben. Weitere Details würden an dieser Stelle zu weit führen und können bei Bedarf im Grundschutzkatalog des BSI nachgelesen werden.[53]

Zusammenfassend ist es durchaus möglich, Anwendungen im Bankenumfeld in eine Cloud auszulagern. Dabei sind Public Clouds als wesentlich kritischer zu betrachten, da bei diesen viel mehr sicherheitskritischen Faktoren berücksichtigt werden müssen. Bei Private und Community Clouds sind die Nutzergruppe, der Standort der Rechenzentren, Zertifizierungen und Sicherheitsstandards wesentlich transparenter und besser zu überblicken. Des Weiteren können hier auch problemlos Audits gefordert und individuelle SLA durchgesetzt werden.

3.3 Gesetzliche Rahmenbedingungen

Im Meldewesen müssen verschiedene Gesetze und Vorschriften beachtet werden. Um später eine gesetzeskonforme Anwendung erstellen zu können, werden diese im folgenden Kapitel thematisiert.

Die Meldung von Großkrediten ist in §13 Großkredite des Kreditwesengesetzes (KWG) geregelt. Dort werden in vier Absätzen die Meldegrenzen, Fristen und weitere Eckpunkte festgelegt.
Im ersten Absatz des §13 wird erklärt, dass „[…]Art, Umfang, Zeitpunkt und Form der Angaben, Übertragungswege und Datenformate der Großkreditstammdatenanzeigen sowie deren Rückmeldungen im Rahmen des Großkreditmeldeverfahrens[…]“[54] durch das Bundesministerium der Finanzen und die Deutsche Bundesbank geregelt werden. Des Weiteren wird dort auf den Artikel 394 der EU Verordnung Nr. 575/2013 verwiesen.[55]

[51] Vgl. Bundesamt für Sicherheit in der Informationstechnik 2014a
[52] Vgl. Bundesamt für Sicherheit in der Informationstechnik 2014b
[53] Vgl. Bundesamt für Sicherheit in der Informationstechnik 2014c
[54] Bundesbank 2015a
[55] Vgl. Bundesbank 2015a

In diesem Artikel werden sowohl der Meldezeitpunkt als auch die Aufbewahrungsfristen beschrieben. Der späteste Meldezeitpunkt ist immer der 15. Geschäftstag eines Kalendermonats. Die Meldungen müssen für das laufende und die zwei Vorjahre aufbewahrt werden.[56]

Durch diese Bestimmungen sind alle relevanten Punkte vordefiniert. Diese müssen bei der Umsetzung eingehalten werden. In dieser Thesis werden speziell die Art, Form und der Übertragungsweg der Meldungen betrachtet. Des Weiteren sind die pünktliche Abgabe und die Speicherung der Daten Randthemen dieser Thesis.

3.4 Implementierung als SaaS

Die Implementierung als SaaS bringt zahlreiche Vorteile mit sich, jedoch entstehen hierdurch auch Abhängigkeiten und Hindernisse, die berücksichtigt werden müssen. Diese werden in diesem Unterkapitel analysiert.

Als Grundlage für eine private Software as a Service wird zuallererst eine Cloud benötigt, welche fähig ist, diese Anwendung auszuführen und auszuliefern. Dazu ist entweder ein IaaS oder ein PaaS nötig, welcher für private Clouds verwendet werden kann. Die Entscheidung, welches Service Modell und welcher Hersteller gewählt wird, hat große Auswirkungen auf die SaaS. Daher ist sinnvoll, einen Vergleich verschiedener Systeme zu machen.

Um eine Anwendung so zu entwickeln, dass alle Vorteile der Cloud genutzt werden können, müssen bestimmte architektonische Maßnahmen getroffen und Regeln eingehalten werden. Diese werden zum größten Teil in Abbildung 6 dargestellt und im Folgenden beschrieben.

[56] Vgl. Bundesbank 2014

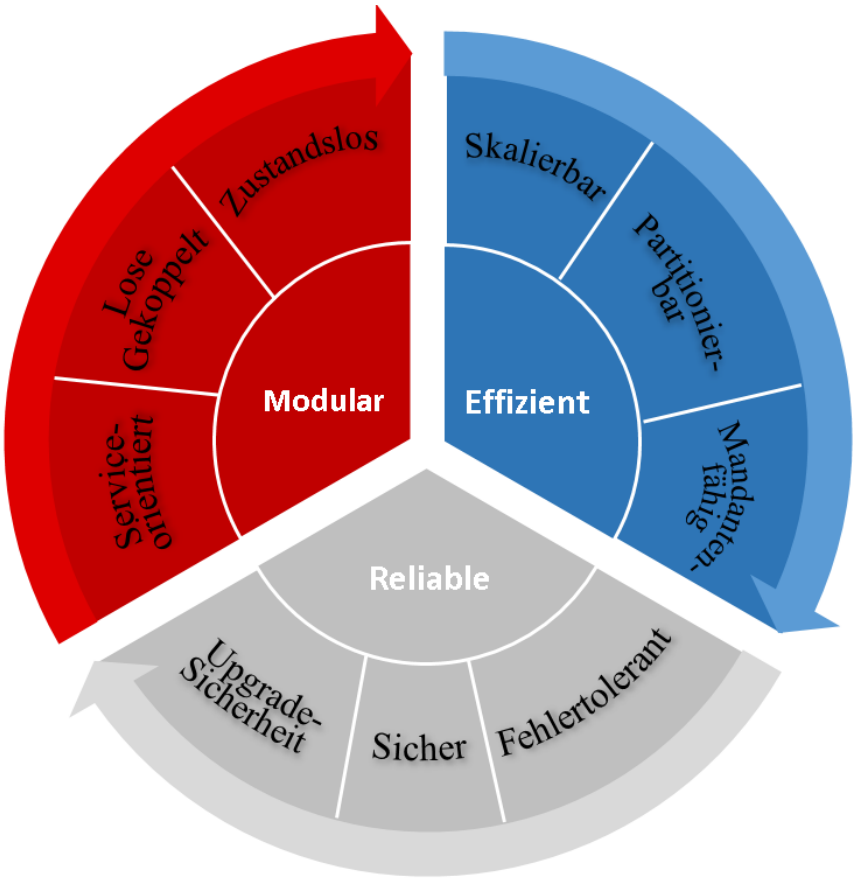

Abbildung 6: SaaS Architektur Charakteristika[57]

Wie zu sehen ist, sind die Charakteristiken von SaaS in drei Bereiche eingeteilt. Einige davon lassen sich auf die serviceorientierte Architektur (SOA) zurückführen, in welchen die SaaS einzuordnen sind. Andere Eigenschaften sind auf das Umfeld, z.B. die Virtualisierung, zurückzuführen. Im ersten Bereich sind Eigenschaften der Modularität dargestellt. Das lässt darauf schließen, dass es sehr wichtig ist, eine SaaS modular aufzubauen. Dies bedeutet zum einen, dass die Anwendung serviceorientiert gestaltet werden muss. Daraus resultiert eine Architektur, welche sich an den Services orientiert. Des Weiteren ist eine lose Kopplung der Module sehr wichtig, da Abhängigkeiten in SaaS zu schlechterer Wartbarkeit und Erweiterbarkeit führen. Als letzter Punkt wird hier die Zustandslosigkeit genannt. Dies ist der wohl wichtigste Punkt der Modularität.[58] Ohne Zustände ist es möglich, die Performanz und Skalierbarkeit zu optimieren. Außerdem kann die Elastizität und Fehlertoleranz verbessert werden. Es können dadurch z.B. zu jedem Zeitpunkt neue Instanzen hinzugefügt werden. Beim

[57] In Anlehnung an: cloudstrategies.biz, 2015
[58] Vgl. Kavis 2014

Absturz einer Instanz müssen keine Zustände wiederhergestellt werden. Es ist dadurch eine schnelle Wiederherstellung möglich.

Dies bedeutet aber auch, dass Dateien nicht auf das Filesystem geschrieben werden dürfen. Damit wäre es nicht möglich, die Instanzen dynamisch zu wechseln. Daher müssen hier andere Lösungen gefunden werden.

Im Bereich Effizienz ist vor allem das Schlüsselwort „Mandantenfähig“ wichtig. Es beschreibt die Fähigkeit eines Systems, mehrere Benutzer handhaben zu können, ohne dass diese Einblicke auf die Daten des jeweils anderen haben. Dies ist nicht nur aus datenschutztechnischen Gründen, sondern auch aus Performanzgründen sehr wichtig.

Die Zuverlässigkeit spielt auch eine wichtige Rolle. Dabei sind, wie in Abbildung 6 zu sehen, die Bereiche Sicherheit und Fehlertoleranz von großer Bedeutung. Fehlertoleranz ist gerade in dem Fall essentiell, wenn virtuelle Instanzen abstürzen und deshalb neue Instanzen gestartet werden müssen. Die Sicherheit ist dahingehend von hoher Bedeutung, dass der Anwender wirklich nur das sieht, was er sehen darf und vertrauliche Daten sicher übertragen werden. Dabei wird oft auf das Need-to-know-Prinzip zurückgegriffen, welches den Datenzugang über ein Rollensystem regelt. Damit wird auch die in Kapitel 3.2 beschriebene Berechtigungsvergabe der MaRisk umgesetzt. Das Übertragen der Daten sollte zusätzlich verschlüsselt stattfinden, um das Abfangen zu verhindern. Dies ist allerdings im Fall einer privaten Verbindung nicht als höchst kritisch anzusehen, da niemand von außerhalb Zugang zu dieser Verbindung hat.

Obwohl durch das Einbinden von Zahlungsstrategien und die Aufnahme neuer Mandanten oft weitere Planung erforderlich ist, wird hier nicht weiter darauf eingegangen, da dies im Sinne der vorliegenden Thesis zu weit führen würde.

Nach der Problemanalyse werden im nächsten Kapitel die Unterschiede zwischen On-Premise und Cloud-Lösungen herausgearbeitet. Abschließend wird die aktuelle Marktsituation analysiert und ein Resümee gezogen.

4 Vergleich von Lösungsansätzen

In diesem Kapitel werden die unterschiedlichen Lösungsansätze verglichen und anschließend die bestehenden Lösungen auf dem Markt analysiert. Um eine klare Abgrenzung zu schaffen, wird bei dem Vergleich der Lösungsansätze vor allem auf die Unterschiede von Cloud-Lösungen zu On-Premise Applikationen eingegangen.

4.1 On-Premise Lösungen

Im Allgemeinen haben lokale Lösungen den Nachteil, dass sie installiert werden müssen. Dafür wird Hardware mit speziellen Spezifikationen benötigt. Außerdem kann es zu Kompatibilitätsproblemen kommen. Des Weiteren ist eine Installation immer ein zusätzlicher Aufwand im Vergleich zu Cloud-Lösungen, da diese entweder manuell oder über einen Installationsserver bzw. ein Skript durchgeführt werden müssen. Auch die Wartung ist in dem Fall viel aufwendiger, da z.B. Updates auf jedem einzelnen System eingespielt werden müssen. Die ganzen Aufgaben sind zudem immer mit Personalaufwand verbunden, wodurch zusätzliche Kosten entstehen.[59]

Dabei wird nicht die Nutzung, sondern der Besitz der Software abgerechnet. Selbst wenn nur vier Mal im Jahr Daten gemeldet werden, muss die Jahreslizenz bezahlt werden.[60]

Der Vorteil im Vergleich zu einer Public Cloud liegt hingegen im Datenschutz und der Datenhoheit. Diese liegen komplett im Unternehmen, dadurch hat dieses vollste Kontrolle über die Verarbeitung und Speicherung der Daten. Gerade bei kritischen Daten, wie jenen im Meldewesen, gibt es sehr strenge Vorschriften betreffend der Datensicherheit und Datenhoheit.

4.2 Cloud-Lösungen

Wie in Kapitel 2.1 beschrieben, gibt es verschiedene Arten von Cloud-Systemen, die unterschiedliche Vor- und Nachteile haben. Da eine Public Cloud aufgrund der Sicherheitsbedenken nur bedingt im Finanzsektor eingesetzt werden kann, werden vor allem die Private und die Community Cloud fokussiert. Somit ist es möglich, die Daten in einem vertrauenswürdigen Rechenzentrum zu hosten und trotzdem die meisten Vorteile einer Cloud nutzen zu können. Einschränkungen gegenüber einer Public Cloud bestehen lediglich in dem Ausmaß der Skalierung und der Kostenreduktion. Dies ist durch den kleineren Nutzerkreis

[59] Vgl. Vossen et al. 2012
[60] Vgl. Vossen et al. 2012

bedingt. Dadurch ist es den Cloud-Hostern oft nicht möglich, die Kosten so weitläufig zu verteilen. Aus demselben Grund bedingt sich auch der Einsatz von weniger Hardware, wodurch die Skalierung begrenzt ist.

Dies vorangestellt, haben Managed Private Cloud Lösungen gegenüber On-Premise Lösungen die Vorteile, dass keine spezielle Hardware in dem Unternehmen vorhanden sein muss. Dadurch entstehen weniger Kosten und die Kompatibilität ist kein Problem mehr, auf das im Unternehmen geachtet werden muss. Updates werden zentral beim Cloud Hoster vorgenommen, wobei auch eine parallele Bereitstellung verschiedener Versionen kein Problem ist.

Einige der in Kapitel 2.1.4 genannten Vorteile sind, bezogen auf den in der Thesis behandelten Anwendungsfall, besonders hervorzuheben. Vorteile entstehen sowohl für die Banken als auch für die Rechenzentren. Die Meldungen sind immer zu einem festen Turnus abzuliefern. In der Zeit zwischen diesen Abgabeterminen wäre es für das Rechenzentrum sinnlos, Ressourcen an die Meldesoftware zu verschwenden. Auf der anderen Seite wäre es für die Banken sehr unvorteilhaft, Gebühren für die Software zu bezahlen. Durch die Cloud-Lösung ist es nun möglich, genau diese Nachteile zu beseitigen, da die Meldesoftware als Service bereitgestellt werden kann und somit eine On-Demand Nutzung möglich ist. Dadurch allokiert das Rechenzentrum nur Ressourcen, wenn diese auch benötigt werden und die Banken zahlen nur dann etwas für die Software, wenn sie diese auch tatsächlich verwenden.

4.3 Bestehende Lösungen

Abschließend werden hier bestehende Reporting Lösungen analysiert und miteinander verglichen. Dabei werden nur Systeme betrachtet, welche es ermöglichen, Meldungen im Rahmen des betrachteten Meldewesens zu generieren.

Programm	BAIS[61]	ABACUS/ DaVinci[62]	Seahorse[63]
Hersteller	BSM GmbH	BESS GmbH	CoreFiling
Meldungen	National und International	National und International	International
Architektur	Fat-Client-Server	Webbasiert Server-Client	SaaS
Datenschnittstelle	Ja	Ja	Nein
Datei-Einreichung	Nein	Ja	Nein
Manuelle Eingabe	Ja	Ja	Mittels XLS-Form
XBRL-Konverter	Ja	Ja	Ja
XBRL-Validation	Ja	Ja	Ja
PDF-Generator	Ja	Ja	Nein
ExtraNet-Schnittstelle	Ja	Ja	Nein
Zahlungsart	Lizenz	Lizenz	On Demand
Voraussetzungen Client	Fat-Client mit JVM	Browser mit JS Adobe Reader und Flashplayer	Browser Microsoft Excel
Server	Batch-Server und Datenbank-Server	Web-Server, ABACUS-Server und DB-Server	-

Tabelle 1: Vergleich von bestehender Meldesoftware

[61] siehe BSM GmbH 2015
[62] siehe BearingPoint Software Solutions GmbH 2015
[63] siehe CoreFiling 2013

Die drei in Tabelle 1 vorgestellten Systeme wurden gewählt, da sie den aktuellen Markt sehr gut repräsentieren. Dabei wurden die ersten beiden Anwendungen als Vertreter für die On-Premise Lösungen gewählt, da sich diese bereits am Markt etabliert und verbreitet haben.

Die Anwendung BAIS von der BSM GmbH hat einen großen Funktionsumfang und stellt bis auf die Datenschnittstelle zu Dateien alle wichtigen technischen Voraussetzungen bereit. Zusätzlich werden hier noch Schnittstellen zu bestehenden Systemen wie SAP und verschiedene Analysemöglichkeiten angeboten. Die Anwendung wird als Java Fat-Client bereitgestellt. Dieser benötigt zur Ausführung einen Fat-Client und eine Java Installation. Des Weiteren werden ein Batch und ein Datenbanken Server benötigt. Somit ist diese Anwendung als klassische lokale Anwendung anzusehen. Dabei sind ein großer Funktionsumfang und ein Abrechnungssystem über feste Lizenzen üblich. Außerdem muss das Unternehmen eigene Hardware bereitstellen und die Wartung selbst übernehmen.

Die Software der BESS GmbH ist eine sehr fortschrittliche On-Premise Lösung, die sich großer Verbreitung erfreut. Diese Software hat den größten Funktionsumfang aller betrachteten Anwendungen und wird als Web-Service bereitgestellt. Hierbei sind bereits unterschiedliche Modelle, wie z.B. eine abgespeckte Light-Version oder eine Outsourcing-Option, berücksichtigt. Durch den Aufbau als Web-Service und die bereits gegebene Outsourcing-Möglichkeit könnte diese Anwendung ohne großen Aufwand in eine Cloud migriert werden.

Als letzte Software wurde Seahorse von CoreFiling als Vertreter der SaaS Anwendungen gewählt. Wie in der Tabelle zu sehen ist, hat diese den geringsten Funktionsumfang. Dies ist aktuell noch häufig bei SaaS-Lösungen zu sehen, da sie noch nicht den gleichen Reifegrad wie On-Premise Lösungen erreicht haben. Mit CoreFiling ist es zwar möglich, Reports zu erstellen, aber durch den geringen Funktionsumfang ist der manuelle Aufwand noch sehr hoch. Diese Lösung benötigt zusätzlich Excel, dadurch geht der Vorteil verloren, dass lediglich ein Browser zum Ausführen der Software benötigt wird. Trotz dieser Probleme sind die Vorteile gegeben, die Software nur On-Demand zu nutzen und auch nur dann dafür zu bezahlen. Updates und aktuelle Taxonomien werden automatisch eingespielt; die Skalierbarkeit ist auch gegeben.

Zusammenfassend kann gesagt werden, dass aktuell eine gemischte Landschaft zwischen verschiedenen On-Premise Lösungen und SaaS Anwendungen vorzufinden ist. Dabei reicht die Spanne der Anwendungen von lokalen Fat-Client Lösungen bis hin zur SaaS Software. Wobei die On-Premise Lösungen hierbei noch klare funktionale Vorteile zeigen. Die interessanteste Beobachtung ist dabei die Annäherung von On-Premise Lösungen zum Cloud-Computing.

Daran ist klar die Tendenz für die nächsten Jahre zu erkennen. Diese geht in Richtung IT-Outsourcing und führt somit letztendlich zur Cloud.

5 Architektur und Anforderungen für RaaS

Dieses Kapitel beschäftigt sich mit dem Aufbau der RaaS Lösung, welche im Rahmen der vorliegenden Thesis entwickelt wird. Hierbei wird zuerst die Grundstruktur der Anwendung dargestellt und erklärt. Anschließend werden verschiedene PaaS verglichen, welche sich für die private Nutzung und Bereitstellung von SaaS eignen. Darauf folgt die Evaluation verschiedener Bibliotheken, die für die Programmierung von großer Bedeutung sind. Die Beschreibung der Testumgebung und eine Anforderungsübersicht schließen dieses Kapitel ab.

5.1 Grundstruktur

In diesem Kapitel wird die Einordnung der Anwendung in den Firmenkontext inklusive der einzelnen Komponenten beschrieben. Im weiteren Verlauf wird auf die verwendeten Komponenten separat eingegangen. Somit ist es möglich, die Grundstruktur und die Zuordnung der Anwendung zu erfassen.

5.1.1 Gesamtübersicht

Anhand von Abbildung 7 werden die Zusammenhänge der unterschiedlichen Systeme erläutert und es wird erklärt, wie sich die Reporting Lösung in dieses Umfeld eingliedert.

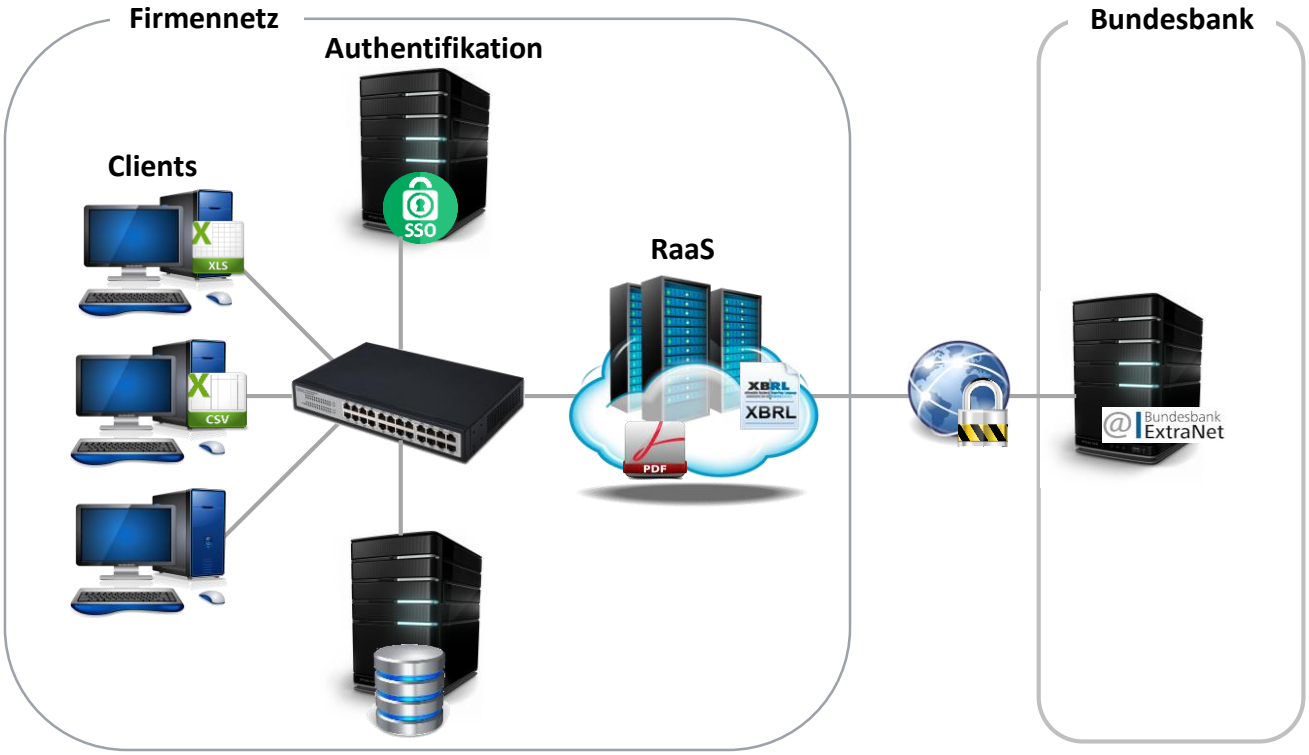

Abbildung 7: Einordnung der Anwendung in den Gesamtkontext[64]

[64] Eigene Darstellung

In der Abbildung wird grundsätzlich zwischen dem Firmennetz, welches eine Anbindung an das Rechenzentrum und somit an die Cloudlösung hat, und dem Internet unterschieden. Das Internet wird lediglich zur Übermittlung von Daten an die Bundesbankschnittstelle zum ExtraNet verwendet. Die restliche Kommunikation findet direkt innerhalb des Firmennetzes und des Rechenzentrums statt.

Das Verwenden der Anwendung wird im Folgenden aus Clientsicht bzw. aus der Sicht der Banken dargestellt. Dabei werden alle Schritte und benötigten Komponenten kurz erläutert.

Zuerst müssen sich die Clients authentifizieren, um zu gewährleisten, dass sie nur auf die Informationen Zugriff haben, wofür die richtigen Berechtigungen vorhanden sind. Um dies so komfortabel wie möglich zu gestalten, soll die Anwendung an eine bestehende Authentifikationsinstanz gekoppelt werden. Mit dem sogenannten Single-Sign-On (SSO) ist es somit möglich, dass sich der Benutzer nur einmal am Rechner anmeldet und die Authentifikation an der Anwendung automatisch erfolgt.

Nachdem der Benutzer sich erfolgreich authentifiziert hat, können Daten via Dateiupload oder mittels einer Datenbankverbindung an die Anwendung weitergegeben werden. Mit der Datenbankverbindung ist es möglich, Datenbanken, welche im Firmen- oder Rechenzentrumsnetz verfügbar sind, direkt an die Anwendung zu koppeln.

Bei der Verarbeitung der Daten wird, wie in Kapitel 3.1 beschrieben, zwischen Stamm- und Betragsdaten unterschieden. Wenn es sich um Betragsdaten handelt, werden diese in XBRL umgewandelt und validiert. Diese XBRL-Dateien können nun automatisch an das ExtraNet der Bundesbank weitergeleitet werden. Dabei müssen die Schritte, wie in Abbildung 8 dargestellt, ausgeführt werden.

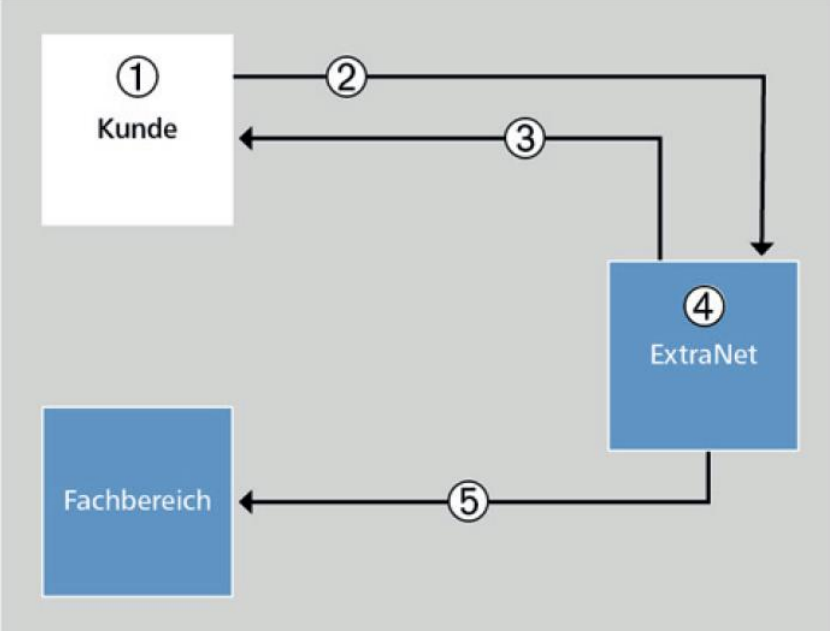

Abbildung 8: Uploadvorgang beim ExtraNet [65]

1. Das Programm, auf Kundenseite, generiert eine XBRL-Datei.
2. Die Datei wird an das ExtraNet übertragen.
3. Das ExtraNet sendet eine Bestätigung des erfolgreichen Uploads, welche dann im Programm angezeigt wird.
4. Das ExtraNet stellt die Dateien zur Abholung für den Fachbereich zur Verfügung.
5. Der Fachbereich holt die Dateien ab.[66]

Falls es sich um Stammdaten handelt, wird das dafür vorgesehene Formular der Bundesbank befüllt und als PDF zum Download bereitgestellt bzw. direkt gedruckt. Diese Fomulare können dann über den Postweg zur Bundesbank gesendet werden.

5.1.2 Komponentenübersicht

In Abbildung 9 werden der Reporting-Service und seine Komponenten dargestellt. Dabei wird das Vorgehen im Anwendungsfluss, also auf der Abbildung von links nach rechts, beschrieben. Um den Service nutzen zu können, müssen diesem als erstes Input-Daten übergeben werden. Dies kann mittels des Uploads von CSV- und XLS-Dateien oder dem Anbinden einer Datenbank geschehen. Anschließend werden die Daten zum Parser weitergeleitet. Hierbei findet bereits eine Unterscheidung der Daten zwischen Betrags- und Stammdaten statt. Diese werden ab diesem Punkt unterschiedlich behandelt. Der Parser filtert nun die nötigen Daten für den jeweiligen Bericht heraus, welche daraufhin in ein Datenobjekt geschrieben werden. Das

[65] entnommen aus Bundesbank 2015b, S. 18
[66] Vgl. Bundesbank 2015b, S. 18

Datenobjekt wird anschließend von dem Validator auf Vollständigkeit und Richtigkeit überprüft. Valide Daten werden in die Datenbank geschrieben, was in nachfolgender Abbildung angedeutet wird. Dies dient zur Kontrolle und erfüllt die Aufbewahrungskriterien, welche bereits in Kapitel 3.3 beschrieben wurden.

An diesem Punkt werden die Stammdaten an den PDF-Builder übergeben. Dieser befüllt das Formular der Bundesbank und erstellt somit ein druckbereites PDF, welches den Meldeansprüchen entspricht. Falls es sich jedoch um Betragsdaten handelt, werden diese vom Konverter in eine XBRL-Datei konvertiert. Anschließend werden diese nochmals anhand der Taxonomie validiert. Abschließend werden die validen XBRL-Dateien, wie oben beschrieben, an die ExtraNet-Schnittstelle gesendet.

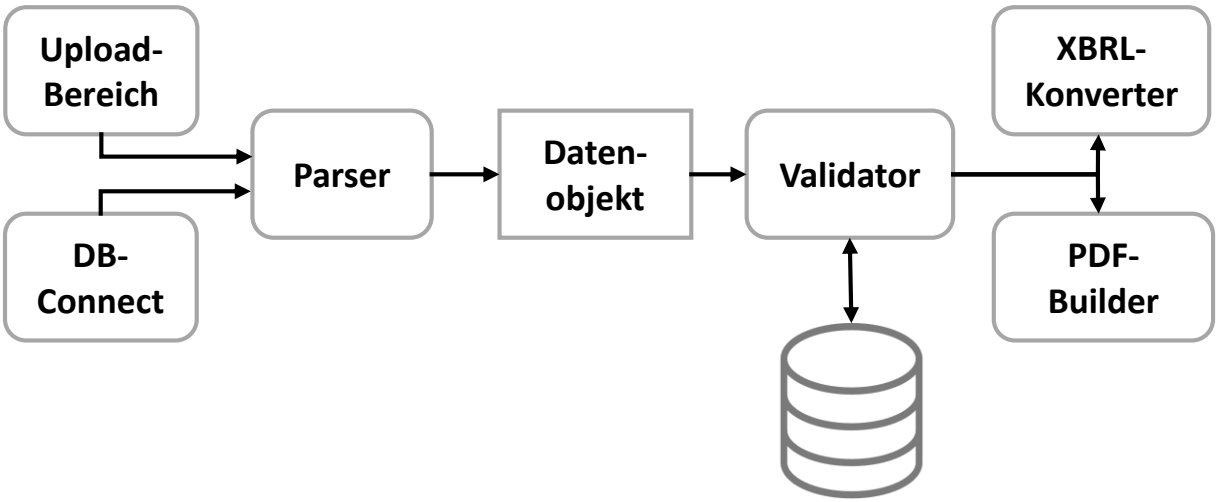

Abbildung 9: Komponenten Diagramm RaaS [67]

Auf die in dieser Abbildung dargestellten Komponenten wird in Kapitel 6 detailliert eingegangen. Dort wird auch deren Entwicklung und Umsetzung beschrieben.

5.2 Vergleich von privaten Cloud-Umgebungen

Wie bereits erwähnt hat die Auswahl der Cloud-Umgebung große Auswirkungen auf die Entwicklung der darauf aufbauenden Anwendungen. Daher wurde zuerst evaluiert, ob ein IaaS oder PaaS eingesetzt werden sollte. Wie bereits genannt bestehen die Unterschiede hauptsächlich in der Abstraktionstiefe.

Zuerst wurde der Ansatz verfolgt, eine IaaS zu verwenden und darauf die Umgebung für die SaaS selbst aufzubauen. Dabei mussten jedoch alle Komponenten, welche für die Ausführung

[67] Eigene Darstellung

der Anwendung benötigt werden, selbst installiert und konfiguriert werden. Dies umfasst unteranderem den Anwendungsserver, die Datenbanken und Management-Tools.

Eine PaaS hingegen bringt diese Komponenten gleich mit und ist zusätzlich darauf ausgelegt, neue Anwendungen zu entwickeln. Da dies genau dem Use-Case entspricht wurde beschlossen, eine PaaS zu verwenden und darauf die SaaS zu entwickeln und zu deployen. Eine PaaS bringt noch weitere Vorteile mit sich. So ist es möglich, diese mit einer IDE zu verbinden und somit die Steuerung der PaaS direkt in der IDE abzuwickeln. Des Weiteren werden Funktionalitäten wie die Versionsverwaltung angeboten und es ist möglich, die oben genannten Komponenten zu managen.

Unabhängig davon, ob eine PaaS oder eine IaaS vorliegt, sollte das zugrundeliegende Cloudsystem privat sein. Dadurch ist es möglich, das System auf eigenen Servern zu installieren. Für diesen POC sollte die gewählte Lösung kostenfrei sein und Java EE unterstützen. Für das Testen bzw. die Umsetzung im Rahmen der Thesis waren die Installationsmöglichkeiten ebenfalls von hoher Bedeutung. Vor allem die Bereitstellung von fertig installierten Images für eine virtuelle Maschine (VM) waren hierbei eine große Erleichterung. Dabei wurden die Systeme zwar teilweise in abgespeckter Form bereitgestellt, was für den POC allerdings völlig ausreichend war. Anhand dieser Kriterien wurde eine Vorauswahl von drei Systemen getroffen

Im Folgenden werden die gewählten Systeme in einer Tabelle verglichen. Anschließend wird anhand festgelegter Kriterien entschieden, welches System sich am besten für die Umsetzung eignet.

	HPE Helion Stackato[68]	Openshift Origin[69]	Cloud Foundry[70]
Hersteller	HPE	Red Hat	Pivotal/VM Ware
Lizenz	HPE End User License	Apache License 2.0	Apache License 2.0
Betriebssystem	Ubuntu 12.04	Fedora, CentOs, RHEL	OpenStack, vSphere, Vagrant
Anwendungs-server	JBoss AS, Tomcat	JBoss AS/Wildfly, Tomcat	Tomcat

[68] siehe HPE Helion Stackato 2016
[69] siehe Red Hat 2016
[70] siehe Cloud Foundry 2016

	HPE Helion Stackato[68]	Openshift Origin[69]	Cloud Foundry[70]
Unterstützte Programmier-sprachen und Frameworks	Clojure Go Java Perl PHP Python Ruby	Java JavaScript .NET Perl PHP Python Ruby	Java JavaScript PHP Python Ruby Scala Go
Frameworks	Node.js Java EE	Node.js Java EE	Node.js Spring
Datenbanken	MySQL PostgreSQL MongoDB Redis	MySQL PostgreSQL MongoDB MS SQL	MySQL PostgreSQL MongoDB Neo4J
Push Via	Eigene Konsole	Git	Eigene Konsole
VM-Images	Ja	Ja	Nein

Tabelle 2: Vergleich von privaten Cloud-Systemen

Wie im tabellarischen Vergleich zu sehen ist, hätte bei Cloud Foundry erst eine Installationsumgebung aufgesetzt werden müssen. Dies hätte im Rahmen der Thesis zu weit geführt, was schließlich zur Entscheidung gegen Cloud Foundry beigetragen hat. Nach diesem Vergleich wurden die VM-Images von Stackato und Openshift getestet. Dabei war der Umgang mit Openshift wesentlich einfacher, da hier schnell neue Applikationen angelegt und via Git mit bestehendem Code befüllt werden konnten. Die Monitoring-Möglichkeiten und der Funktionsumfang sind bei Stackato jedoch weiter ausgereift.[71] Für diesen POC sind diese allerdings nicht von ausschlaggebender Bedeutung.

Aufgrund der Einfachheit in der Anwendungserzeugung wurde Openshift Origins von Red Hat als PaaS für diese Thesis ausgewählt.

5.3 Vergleich von XML- und XBRL-Bibliotheken

Zu den Kernelementen der Anwendung zählen der Konverter und der XBRL-Validator. Deren Aufgabe ist, das Instanz-Dokument zu erstellen und dieses anschließend mit der entsprechenden Taxonomie zu validieren. Um diese Komponenten umzusetzen, wurden XML-

[71] Vgl. HPE Helion Stackato 2016

oder XBRL-Bibliotheken benötigt. Dabei bietet eine XBRL-Bibliothek dahingehend Vorteile gegenüber einer XML-Bibliothek, dass damit das Validieren wesentlich einfacher ist. Dies ist dadurch bedingt, dass die XBRL-Librarys speziell dafür entwickelt wurden und deshalb nicht nur die Syntax, sondern auch die Semantik verstehen.[72] Um die Kosten für diesen POC gering zu halten, wurden nur Open Source oder Freeware APIs überprüft. Dabei wurde schnell klar, dass es nur sehr wenige kostenlose Projekte oder Bibliotheken im XBRL-Bereich gibt. Bei den bestehenden Projekten fehlt es zusätzlich oft an Aktualität und an ausreichender Dokumentation.[73]

Daher fiel die Entscheidung in Bezug auf das Instanz-Dokument auf den Standard XML-Parser von Java. Dieser besitzt den nötigen Funktionsumfang für diese Aufgabe und ist zusätzlich sehr gut dokumentiert.

Für die Validation wurde keine kostenlose Bibliothek gefunden, welche in dem gegebenen Umfeld eingesetzt werden kann. Dafür muss also eine eigene Lösung entwickelt werden.

5.4 Vergleich von PDF-Bibliotheken

Um automatisiert ein PDF mit den Stammdaten zu generieren, kann entweder ein eigenes PDF erstellt oder das Formular der Bundesbank ausgefüllt werden. Da beim Erstellen auf die Vorgaben der Bundesbank geachtet werden muss und das Generieren im Gesamten einen höheren Aufwand darstellt, wurde Variante zwei gewählt. Auch bei Erneuerungen des Formulars oder Erweiterung der Anwendung ist das Einbinden neuer Formulare effizienter als die Erstellung eigener Formulare. Für den Umgang mit PDF-Dateien stehen sehr viele Bibliotheken für Java zur Verfügung. Viele davon sind aber nicht in der Lage, PDF-Formulare zu befüllen. Die zwei gängigsten Bibliotheken, welche auch in der Lage sind Formulare zu befüllen, sind iText und Apache PDFBox.

Da iText ab Version 5 unter der Affero General Public License veröffentlicht wurde, müssen Lizenzen für Closed Source gekauft werden. Die PDFBox von Apache ist unter der Apache License veröffentlicht und wird zudem noch aktiv weiterentwickelt, daher fiel die Entscheidung auf diese Bibliothek.

[72] Vgl. Hoffman und Watson 2010
[73] Vgl. Hoffman und Watson 2010

5.5 Aufbau einer Testumgebung

Um die Anwendungen testen bzw. entwickeln zu können, muss zuerst eine geeignete Umgebung eingerichtet werden. Da die Endanwendung eine Software as a Service Lösung sein soll, die in einer privaten Cloud läuft, musste eine PaaS aufgesetzt werden, um dort die Applikation zu integrieren und zu testen. Des Weiteren musste eine Datenbank installiert werden, um die Anbindung an eine Datenbank bzw. die Konvertierung von Daten direkt aus einer Datenbank, zu testen. Datenbanken können mittels der Java Database Connectivity (JDBC) an eine Java-Anwendung angebunden werden. Für die gängigen Datenbanken existiert auch ein passender JDBC-Treiber, um die Anbindung zu realisieren. Die MySQL-Datenbank wurde gewählt, da sie eine der verbreitetesten und bekanntesten Open Source Datenbanken ist.[74] Dadurch kann sie gut als Referenz eingesetzt werden.

5.5.1 Installation der PaaS

Wie in Kapitel 5.3 beschrieben wurde die PaaS-Lösung Openshift Origin von Red Hat gewählt. Diese kann entweder auf einem Linux installiert oder direkt in einer virtuellen Maschine oder einem Container, wie z.B. Docker, ausgeführt werden. Für den Testaufbau wurde das fertig installierten VM-Ware Image gewählt, da hierbei wenig konfiguriert werden muss. Diese Lösung ist somit am einfachsten umzusetzen. Des Weiteren wird eine PaaS bei der Verwendung oberhalb einer IaaS ebenfalls in einer virtuellen Instanz ausgeführt. Somit sind hierbei auch realitätsnahe Bedingungen gegeben.

Nach dem Starten der virtuellen Maschine, welche eine Openshift Origin Installation auf CentOs-Basis enthält, kann direkt via Web-Interface auf OpenShift zugegriffen werden. Dabei sind einige Cartridges bereits vorinstalliert. Ein Cartridge ist ein Modul, welches alle nötigen Komponenten zum Entwickeln, Deployen und Ausführen einer Anwendung enthält. Vorinstalliert waren php-, phyton- und ruby-Cartridges.

Um eine Java EE Anwendung mit dieser Openshift Version entwickeln zu können, mussten noch weitere Cartridges installiert werden. Diese benötigten im Hintergrund wiederum Pakete, welche für das Kompilieren und Ausführen nötig sind. In diesem Fall wurde ein Java Developer Kit (JDK) zum Kompilieren und ein Anwendungsserver für das Deployen und Ausführen der Webanwendung benötigt. Beim Anwendungsserver wurde sich für JBoss entschieden, da dieser zum gegebenen Zeitpunkt am besten in Openshift integriert war und alle weiteren

[74] Vgl. Schicker 2014, S. 12

Anforderungen erfüllt hat.

Zum Installieren der Komponenten mussten mit dem Paketverwaltungstool yum die JBoss- und Maven-Pakete geholt werden. Dabei ist das JDK bereits als Abhängigkeit enthalten. Die Download-Server waren teilweise sehr langsam, daher musste der Timeout von yum erhöht werden. Sonst konnte es vorkommen, dass die Downloads automatisch abgebrochen wurden. Nachdem die Pakete installiert waren, konnten die Cartridges via yum zu Openshift hinzugefügt werden. Nach diesem Schritt waren die neuen Module für Java im Web-Interface verfügbar und konnten verwendet werden. Nachdem das Java-Modul ausgewählt wurde, kann die Anwendung, wie in Abbildung 10 und Abbildung 11 zu sehen, konfiguriert werden.

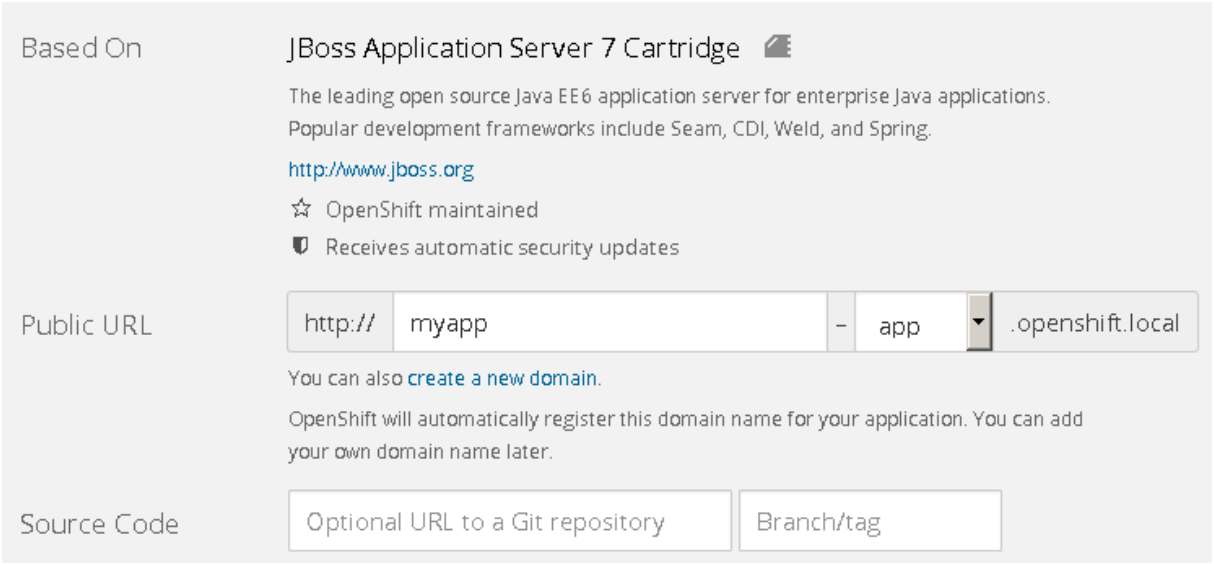

Abbildung 10: Oberer Teil des App Konfigurators von Openshift

Im Fall einer JBoss-Anwendung können im oberen Teil, welcher in Abbildung 10 dargestellt ist, das Cartridge eingesehen, die URL und ein bestehendes Git-Repository eingegeben werden. In dieser Maske ist es zudem möglich Domains anzulegen und zu bestimmen. Nachdem der *Create Application* Button betätigt wurde, wird ein Git-Repository angelegt und ein Projekt mit den entsprechenden Komponenten erstellt. Wenn ein bestehendes Git-Repository angegeben wurde, wird der Inhalt daraus automatisch kopiert. Andernfalls wird eine Beispielanwendung angelegt. Somit ist die Möglichkeit geboten, bestehende Anwendungen sehr schnell zu deployen und eventuell um einen Load-Balancer oder andere Komponenten zu erweitern.

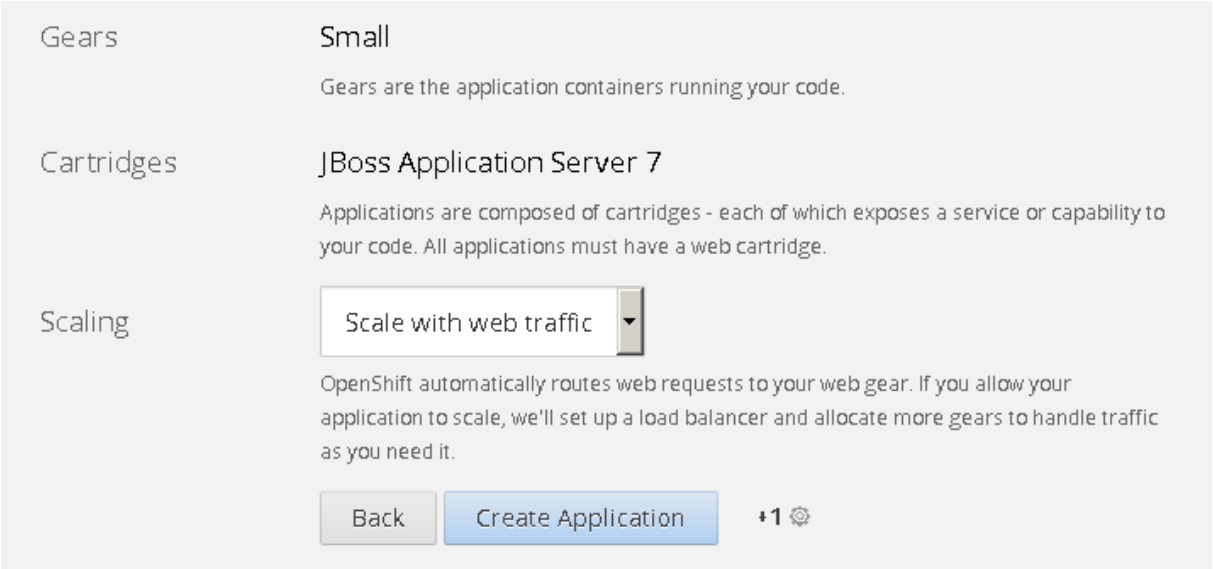

Abbildung 11: Unterer Teil des App Konfigurators von OpenShift

Im unteren Teil des App Konfigurators können, wie in Abbildung 11 zu sehen ist, die Cloud relevanten Einstellungen vorgenommen werden. In diesem Fall können die Größe der Instanz, sowie das Load-Balancing eingestellt werden.

Nachdem die Anwendung erzeugt wurde, wird eine Übersichtsseite angezeigt. Auf dieser ist es, wie in Abbildung 12 zu sehen, möglich, Einstellungen anzupassen oder Komponenten hinzuzufügen. Des Weiteren wird auf dieser Seite ein Link zu dem erstellten Git-Repository bereitgestellt. Über diesen kann mittels Secure Shell (SSH) eine verschlüsselte Verbindung aufgebaut werden. Dafür wird ein asymmetrisches Schlüsselpaar benötigt. Der Public Key muss bei OpenShift hinterlegt werden. Der private Schlüssel muss auf dem Host zur Verfügung stehen, um eine erfolgreiche Authentifizierung durchzuführen. Nachdem diese durchgeführt wurde, kann das Repository geklont, commitet und gepusht werden.

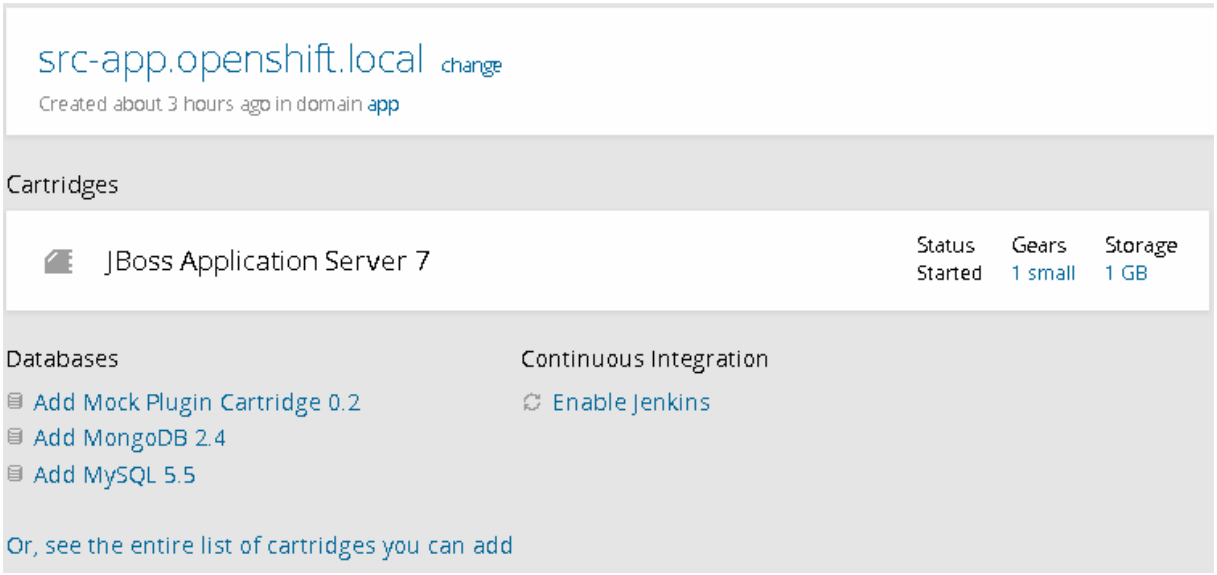

Abbildung 12: Übersichtsseite der Anwendung in OpenShift

5.5.2 Vorbereitung der Daten und Dienste

Um die Anwendung entsprechend testen zu können, mussten vorab Daten generiert und Dienste konfiguriert werden.

Bei der Datengenerierung waren Betrags- und Stammdaten nötig. Um alle Funktionalitäten testen zu können, wurden sowohl Tabellen und Views, als auch XLS-, XLSX- und CSV-Dateien erstellt. Für die Testzwecke mussten sowohl valide- sowie nicht-valide-Daten erzeugt werden. Die Export- und Konvertierungsmöglichkeiten waren hierbei sehr hilfreich. So war es z.B. möglich, die Daten aus den Tabellen als CSV zu exportieren. Diese konnten dann wiederum in XLS- und XLSX-Daten umgewandelt werden. In Tabelle 3 werden die generierten Daten beschrieben.

Nummer	Typ	Valide	Beschreibung
BD-001	LE1	Ja	Valide und vollständige Datensätze der Meldung LE1
BD-002	LE1	Nein	Nicht-valide und vollständige Datensätze der Meldung LE1
BD-003	LE1	Nein	Valide und unvollständige Datensätze der Meldung LE1
BD-004	LE Limits	Ja	Valide und vollständige Datensätze der Meldung LE1

BD-005	LE Limits	Nein	Nicht-valide und vollständige Datensätze der Meldung LE Limits
BD-006	LE Limits	Nein	Valide und unvollständige Datensätze der Meldung LE Limits
SD-001	Stamm-daten	Ja	Valide und vollständige Datensätze für die Verarbeitung als Stammdaten
SD-002	Stamm-daten	Nein	Nicht-valide und vollständige Datensätze für die Verarbeitung als Stammdaten
SD-003	Stamm-daten	Nein	Valide und unvollständige Datensätze für die Verarbeitung als Stammdaten

Tabelle 3: Beschreibung der Testdaten

Die Präfixe der Nummern geben an, ob es Stamm- oder Betragsdaten sind. Der Typ spezifiziert genauer, um welche Daten es sich handelt. Die Spalte „Valide“ gibt Auskunft über die Datenvalidität. In der Beschreibung werden detaillierte Informationen zu den Daten dargestellt.

Zur Vereinfachung der Traceability werden bei den Testfällen in Kapitel 0 die jeweils verwendeten Testdaten angegeben. Weder die Testdaten noch die Tests besitzen dabei den Anspruch auf Vollständigkeit.

Außerdem wurden Datenbanken installiert, konfiguriert und befüllt, um diese Schnittstelle testen zu können. Exemplarisch wird dieser Vorgang an der MySQL-Datenbank beschrieben.

Die Installation der Datenbank erfolgt über den Setup-Wizard. Um anschließend die Konfiguration mit einem grafischen Tool vorzunehmen, wurde das Verwaltungsprogramm Workbench installiert. Bei diesem konnte per Knopfdruck eine Datenbank erzeugt werden. Nachdem die Datenbank angelegt wurde, konnten dort Tabellen erstellt werden. Bei der Erstellung der Tabellen wurden die Testdaten aus Tabelle 3 als Orientierung verwendet. Da auch Views unterstützt werden sollen, mussten diese ebenfalls erzeugt werden. Die Views bauen dabei auf den bereits angelegten Tabellen auf. Nachdem die Tabellen und Views erstellt wurden, konnten diese mit den Testdaten befüllt werden.
Um realistische Testbedingungen zu erzeugen wurde ein spezieller User mit eingeschränkten Rechten für die Verbindung angelegt. Mit diesem User war es lediglich möglich, eine Verbindung mit der Datenbank aufzubauen und Daten abzufragen. Eine Einschränkung auf bestimmte Tabellen wurde in diesem Fall nicht vorgenommen, da die Datenbank ausschließlich die Testdaten enthält.

5.6 Übersicht der Anforderungen

Dieses abschließende Kapitel gibt einen Überblick über die herausgearbeiteten Anforderungen. Jene werden in Tabelle 4 dargestellt und in den entsprechenden Kapiteln, bei denen sie zur Anwendung kommen, referenziert.

Nummer	Anforderungsbeschreibung
FS-001	Das System ist in der Lage, Daten von verschiedenen Schnittstellen zu lesen, insbesondere CSV-Upload, Excel-Upload und gängige Datenbanken (MySQL, Oracle, H2).
FS-002	Das System validiert Daten, die es über die Schnittstellen einliest.
FS-003	Das System befüllt, in einer Datenbank bereitgestellte, PDF-Formulare mit den eingelesenen und validierten Daten.
FS-004	Das System konvertiert eingelesene Daten in das Dateiformat XBRL und validiert diese Dateien anhand vorgegebener Taxonomien.
FS-005	Das System ist in der Lage, validierte XBRL-Dateien an das ExtraNet zu senden.
FS-006	Das System bietet die Möglichkeit, sich an einem bestehenden Authentifizierungssystem zu authentifizieren.
NFS-001	Das System wird als Software as a Service bereitgestellt.
NFS-001.1	Das System ist dadurch skalierbar.

Tabelle 4: Anforderungsübersicht

Da es sich um einen POC handelt, wird größtenteils auf nicht-funktionale Anforderungen verzichtet. Natürlich sind Faktoren wie Zuverlässigkeit, Benutzbarkeit, Änderbarkeit, Flexibilität und Sicherheit trotzdem sehr wichtig.

Durch die in diesem Kapitel vorbereitete Testumgebung und den herausgearbeiteten Anforderungen kann der POC umgesetzt werden. Die Umsetzung wird im folgenden Kapitel beschrieben.

6 Umsetzung des Lösungskonzeptes für RaaS

In diesem Kapitel wird die Umsetzung der Anforderungen erläutert. Dabei wird auf die Erkenntnisse, welche im vorhergehenden Kapitel gewonnen wurden, aufgebaut. Auch die Testumgebung, welche dort beschrieben wurde, wird als gegeben betrachtet. Die Gliederung der Unterkapitel orientiert sich größtenteils an der Komponentenübersicht aus Kapitel 5.1.2, um eine klare Trennung der Komponenten und der dazu umzusetzenden Implementierungsschritte zu erreichen.
Abschließend werden die Testverfahren und Testmethoden beschrieben, welche eingesetzt wurden, um die Qualität der Implementierung zu gewährleisten.

6.1 Authentifikation und SSO

Eine Authentifikation ist wichtig, da nur so sichergestellt werden kann, dass ein berechtigter Sachbearbeiter Zugriff auf die Applikation und die damit verbundenen Daten bekommt. Des Weiteren kann nur über eine Authentifikation mit Rollen und Gruppen das Need-to-know-Prinzip umgesetzt werden. Zusätzlich entsteht durch das Login bzw. die Authentifikation auch die Möglichkeit der Individualisierung. So können Daten hinterlegt werden, die auf den jeweiligen Sachbearbeiter bezogen sind. Diese sind bei der Registrierung einzutragen und können über ein Konfigurationsmenü angepasst werden. Sie werden für die Headerinformationen der Meldungen und für die Protokollierung benötigt. Außerdem können konfigurierbare Elemente, wie z.B. Datenbankverbindungen gespeichert werden. Somit müssen diese nicht bei jedem Meldevorgang neu eingerichtet werden.

Durch das Single-Sign-On muss der Benutzer sich nicht extra an der Anwendung authentifizieren. Dies erleichtert den Zugang zu der Anwendung und erhöht die Sicherheit im Unternehmen. Für die Implementierung von SSO stehen viele unterschiedliche Frameworks und Authentifizierungsmethoden zur Verfügung.[75] Daher ist es schwer, eine Lösung zu implementieren, welche in jedes Firmennetzwerk eingebunden werden kann. Oft haben Firmen auch bereits ein SSO-System im Einsatz, welches meist eine API bereitstellt, um andere Anwendungen daran anzubinden. Die Verwendung dieser oder der Einsatz eines SSO-Frameworks wird empfohlen. Dadurch können im Vergleich zu selbst entwickelten Lösungen wesentlich höhere Sicherheitsstandards und ein breiteres Spektrum an Technologien unterstützt werden.

[75] Vgl. Institute of Electrical and Electronics Engineers et al. 2011

Die in dieser Thesis entwickelte Anwendung erfüllt alle nötigen Kriterien, welche für die Anbindung an ein bestehendes SSO-System nötig sind. Auf die Implementierung von SSO wurde allerdings im Rahmen des POC verzichtet, da dies neben den genannten Punkten zu einem erheblichen Mehraufwand für die Testumgebung geführt hätte. Die Anforderung **FS-006** ist daher nur Teilweise erfüllt.

6.2 Grafische Oberflächen

In diesem Unterkapitel werden lediglich die grafischen Oberflächen und die damit verbundene Technologie beschrieben. Eine detaillierte Beschreibung der verwendeten Methoden, Datenobjekten und Techniken im Hintergrund wird in den folgenden Unterkapiteln gegeben. Als grundsätzliche Technologie wird für den Aufbau der grafischen Oberfläche Java Server Faces (JSF) verwendet, welche mit XHTML-Dateien arbeitet. Mit diesen ist es möglich, innerhalb der grafischen Web-Oberfläche auf die Methoden und Attribute der Java-Klassen im Hintergrund zuzugreifen. Um ein einheitliches Look & Feel zu erzeugen, wird darauf aufbauend das UI-Framework Primefaces verwendet. Dieses Framework bietet viele grafische Komponenten mit konfigurierbaren Modi.

Um das im vorherigen Unterkapitel beschriebene Konzept umsetzen zu können, müssen sich die Benutzer zuerst bei der Anwendung registrieren. Dafür steht die in Abbildung 13 dargestellte Maske zur Verfügung.

Abbildung 13: Registrierungsmaske

Wie in der Abbildung zu sehen, müssen hier personenbezogene Daten eingegeben werden. Diese sind essenzielle Bestandteile der Meldungen. Durch das einmalige Eintragen der Daten bei der Registrierung werden diese bei jeder Meldung automatisch eingefügt. Sollten sich die

Daten ändern, können diese über eine Änderungsmaske angepasst werden. Die Änderungsmaske ist optisch identisch mit der oben abgebildeten Maske und wird daher nicht separat abgebildet. Auf technischer Seite wird ein Formular mit Input-Feldern und Labeln verwendet. Beim Klick auf den *registrieren*-Button werden die Eingaben direkt auf Plausibilität und Vollständigkeit überprüft und anschließend in die Datenbank gespeichert.

Nach erfolgreicher Registrierung und Authentifizierung erscheint beim Start die in Abbildung 14 dargestellte Start-Maske. In dieser Ansicht können Datentyp und Verarbeitungsart gewählt werden. Dabei ist eine Wahl zwischen Stamm- und Betragsdaten, sowie zwischen Dateiupload und Datenbankverbindung möglich. Diese werden mittels einer *oneButton*-Komponente umgesetzt. Bei diesem Element kann nur eine Option ausgewählt werden. Nachdem beide Angaben gemacht wurden, wird der Verarbeitungsprozess mit dem *Weiter*-Button gestartet.

Abbildung 14: Startbildschirm

Um die Reporting-Daten der Anwendung übergeben zu können, ist es, wie in Anforderung **FS-001** definiert, nötig, verschiedene Schnittstellen zu den Datenpools der Benutzer bereitzustellen. Da die Daten meist in Dateien oder Datenbanken gespeichert sind, müssen hierfür Schnittstellen bereitgestellt werden. Da es sich bei den Nutzern der Anwendung um Endanwender handelt, muss hier eine möglichst simple grafische Oberfläche für diese Schnittstellen bereitgestellt werden. Dabei soll der Dateiupload die Möglichkeit bieten, die Daten für jeden Report-Termin einzeln hochzuladen und zu verarbeiten. Die Datenbankschnittstelle soll dahingegen einmalig konfiguriert werden. Für die nachfolgenden Reports kann auf die gespeicherten Schnittstellen zugegriffen werden.

Für diesen POC wurde die Datenbank MySQL und die Datenformate XSLX und CSV als Input gewählt, da diese sehr weit verbreitet sind. Die Architektur des Services wurde allerdings erweiterbar gestaltet, so können neue Formate oder andere Datenbanken ohne Probleme hinzugefügt werden.

Für das Hochladen der Daten wird als Erstes ein File-Chooser oder ähnliches benötigt. Wie der Name schon sagt, ist es für den Nutzer damit möglich, die Datei auszuwählen. Für den POC wurde der *fileUpload* Mechanismus von PrimeFaces im *advanced Mode* genutzt, welcher in Abbildung 15 zu sehen ist.

Abbildung 15: Upload Bereich

Damit ist es möglich, Dateien direkt per Drag & Drop oder klassisch über die Durchsuchen-Funktion auszuwählen. Dabei können verschiedene Einschränkungen bezüglich Dateityps und Dateigröße vorgenommen werden. Dies verhindert das Hochladen falscher Dateien und ist eine erste Barriere für Anwenderfehler. In diesem Fall wurde die Auswahl auf XLS-, XLSX- und CSV-Dateien eingeschränkt. Für die Größe wurde ein Limit von 10 MB festgelegt. Hierbei ist eventuell eine Anpassung an die Anforderungen der verschiedenen Banken nötig. Dabei ist es auch möglich, mehrere Dateien auf einmal auszuwählen und hochzuladen.

Durch das Drücken des *Upload*-Buttons wird ein Event ausgelöst, welches von der *handleFileUpload*-Methode angenommen wird. Dort werden die hochgeladenen Dateien entsprechend ihres Typs an spezifische Parser-Klassen weitergegeben. Diese Parser verarbeiten die Daten aus den Dateien. Details hierzu werden in dem nächsten Unterkapitel erläutert.

Das Verbinden mit der Datenbank hingegen ist ein mehrstufiger Prozess, welcher aber, wie bereits beschrieben, nur einmalig für jede Datenquelle ausgeführt werden muss. Zuerst müssen die verbindungsrelevanten Daten eingegeben werden. Hierzu wurde das Formular, welches in Abbildung 16 zu sehen ist, erstellt. Die Verbindung wird intern über JDBC hergestellt. Dabei sind alle Informationen nötig, welche durch das Formular abgefragt werden. Hierbei ist es sehr wichtig, dass alle Daten richtig eingetragen werden, da sonst keine Verbindung erzeugt werden kann. Deshalb wird die Eingabe direkt auf Plausibilität geprüft. Dabei werden leere Felder oder fehlerhafte Eingaben direkt erkannt und der Benutzer wird darauf hingewiesen, dass etwas geändert werden muss. Wurden alle Daten eingegeben, ist ein Test der Verbindung möglich. Mit dem Button „Lade Tabellen“ ist daneben das direkte voranschreiten zum nächsten Schritt des Prozesses möglich.

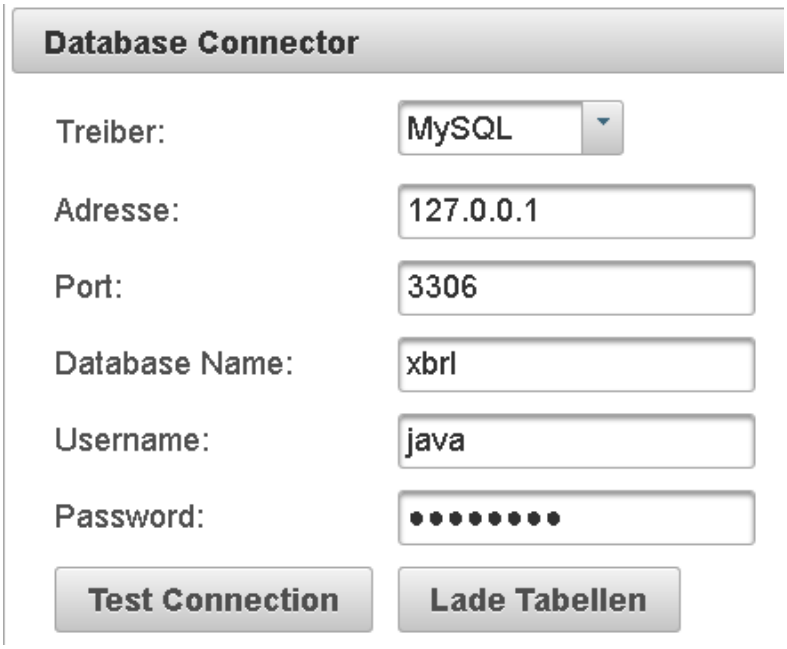

Abbildung 16: Formular der Datenbankschnittstelle[76]

Im zweiten Schritt des Prozesses ist es möglich, die Tabellen oder Views der Datenbank auszuwählen, welche die gewünschten Daten enthalten. Dabei werden, wie in Abbildung 17 zu sehen, alle Tabellen und Views, für die der User Berechtigungen hat, in einer *selectOneListbox* angezeigt.

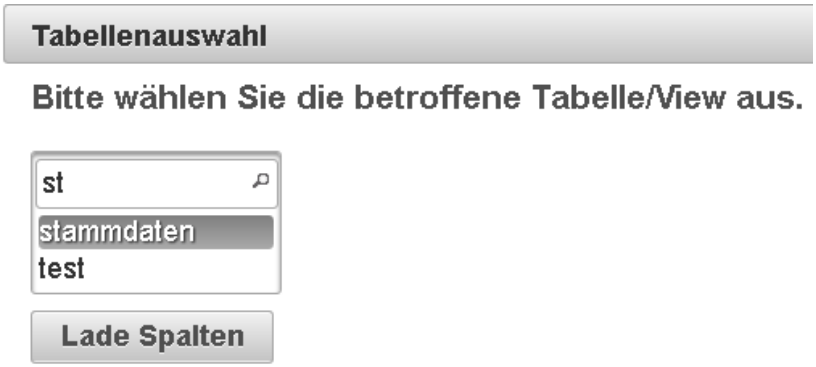

Abbildung 17: Tabellen und View Auswahl der Datenbankschnittstelle [77]

Hierbei kann nur ein Element ausgewählt werden. Mittels der Suchmaske ist es möglich, einen Filter über die Tabellen und Views zu legen. Dadurch gestaltet sich das Suchen der Tabellen wesentlich effizienter.

Aggregationen, Mehrfachauswahl und Einschränkungen auf die Tabellen sind hier bewusst nicht möglich. Diese müssen im Voraus gemacht und als Views bereitgestellt werden. Dies verhindert nicht nur die Steigerung der Komplexität, sondern ist sicherheitstechnisch die

[76] Screenshot des RaaS
[77] Screenshot des RaaS

wesentlich bessere Lösung. Nachdem die gewünschten Tabellen und Views ausgewählt wurden, kann mittels des Buttons „Lade Spalten“ zum Letzten Prozessschritt gelangt werden.

Abschließend müssen durch den Anwender die gewünschten Spalten aus den Tabellen ausgewählt werden. Wie in Abbildung 18 zu sehen, wird hier eine *pickList* eingesetzt. Bei dieser ist es möglich, mittels der Checkboxen eine Mehrfachauswahl zu treffen, einzelne Elemente zu verschieben oder alle Spalten auszuwählen. Ebenfalls kann hier mittels der Suchmaske eine Filterung vorgenommen werden.

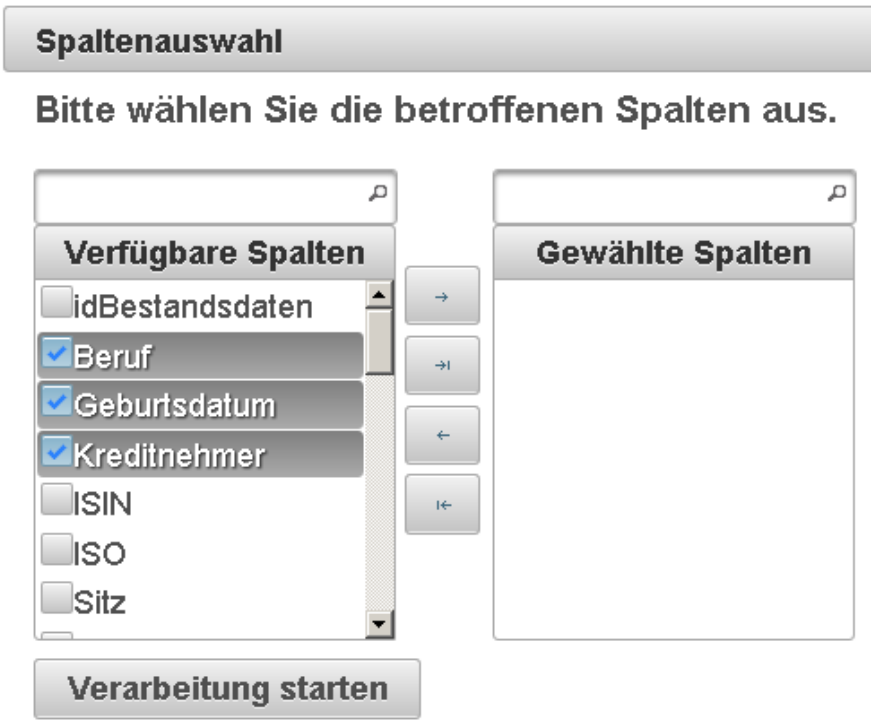

Abbildung 18: Spaltenauswahl der Datenbankschnittstelle[78]

Nachdem die Auswahl der Spalten abgeschlossen wurde, kann mit dem Button „Verarbeitung starten“ die Verarbeitung gestartet werden. Die Daten werden anhand der Namen in die entsprechenden Felder der Datenobjekte eingefügt.

Wurden die Daten eingelesen und verarbeitet, werden sie in einer Kontrollansicht dargestellt. Dort kann eine manuelle Überprüfung vorgenommen werden. Diese Ansicht erlaubt es auch, die Dateien herunterzuladen und zu versenden. Dabei gibt es unterschiedliche Ansichten für die Stamm- und Betragsdaten. Bei den Stammdaten werden die fertiggestellten PDF-Dateien anzeigt. Dies ist in Abbildung 19 ersichtlich.

[78] Screenshot des RaaS

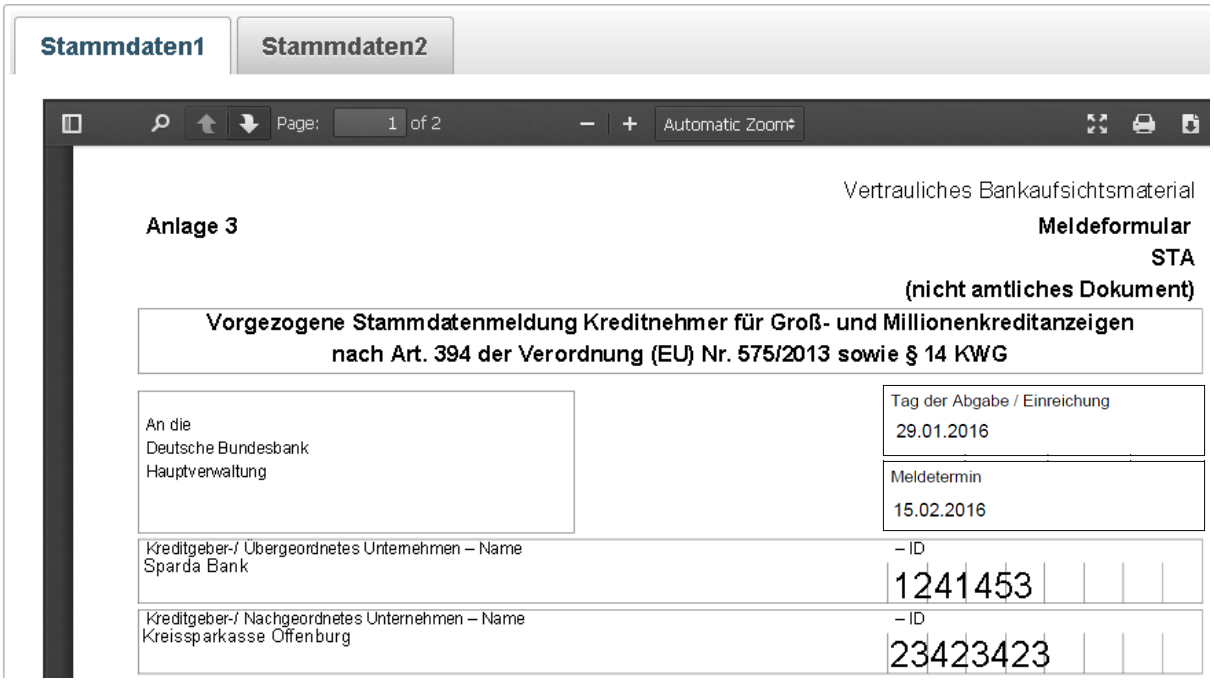

Abbildung 19: Beispiel des PDF-Viewers

In dieser Ansicht können die ausgefüllten PDF-Dateien auf Richtigkeit untersucht werden. Dabei wird die *tabView*-Komponente verwendet, um mehrere Dateien auf einer Seite anzeigen zu können. Innerhalb dieser Tabs kommt der *documentViewer* zum Einsatz, welcher Möglichkeiten zum Drucken und Downloaden der Dateien bietet.

Bei der Kontrollansicht der Betragsdaten werden, wie in Abbildung 20 dargestellt, die versandbereiten XBRL-Dateien anzeigt.

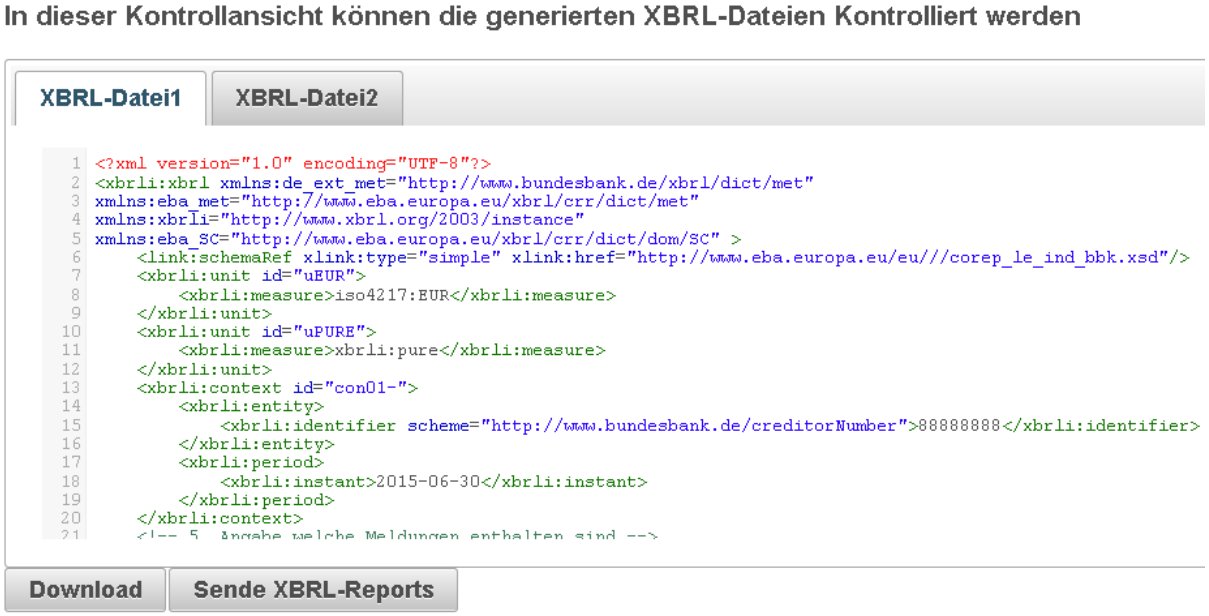

Abbildung 20: XBRL-Viewer

Hierbei wird ebenfalls die *tabView*-Komponente eingesetzt. Die Ansichten unterscheiden sich allerdings in der Anzeigekomponente und den Verarbeitungsoptionen. So wird beim XBRL-Viewer ein *codeMirror* zur Anzeige verwendet. Außerdem wurde an dieser Stelle die Methode hinterlegt, die das Senden der XBRL-Dateien an das ExtraNet ermöglicht. Das Downloaden ist zusätzlich mittels der Komponente *fileDownload* realisiert.

6.3 Datenobjekte und Parser

In diesem Kapitel werden die Datenobjekte beschrieben, welche für die Abbildung der Meldedaten benötigt werden. Anschließend wird erläutert, wie die Datenbefüllung durch die unterschiedlichen Parser vorgenommen wird.

Bei den Datenobjekten findet eine Unterscheidung zwischen Stamm- und Betragsdaten statt. Um die Stammdaten abbilden zu können, müssen lediglich die Felder eines PDF-Formulars in ein Datenobjekt übernommen werden. Somit enthält ein Stammdatenobjekt genau die Inhalte für ein Formular. Dabei werden lediglich String Felder eingesetzt, da die Felder des PDF-Formulars nur Strings annehmen können. Die Prüfung auf Korrektheit und Plausibilität findet bereits während des Parsens mit dem Validator statt. Näheres dazu wird in Kapitel 6.6.1 erläutert.

Bei den Betragsdaten müssen die Daten eines Instanz-Dokuments abgebildet werden. Wie bereits in den Grundlagen erläutert, ist die Meldung modular aufgebaut und besteht somit aus mehreren Teilen. Insgesamt gibt es die Teilmeldungen LE1-LE5 und LE Limits. Diese können zum Teil nur im Verbund gemeldet werden, teilweise aber auch einzeln. Daher muss ein Datenobjekt in der Lage sein, mehrere Teilmeldungen zu enthalten. Des Weiteren muss in dem gegebenen Meldekontext jedes Instanz-Dokument die Felder des C00.01 und die Headerinformationen der Bundesbank enthalten. Für die Umsetzung dieser Anforderungen wurde für jeden Meldevordruck (LE1-LE5 und LE Limits) eine eigene Klasse erstellt. Das Betragsdatenobjekt selbst enthält, wie in Listing 2 zu sehen, die Headerinformationen und die Felder des C00.01. Zusätzlich ist es hier möglich, Instanzen der Meldeklassen einzubinden.

```
public class Betragsdaten {
          //Header Informationen
          private String IdentnummerAbs;
          private String IdenttypeAbs;
          private String IdentnummerMeld;
          private String IdenttypeMeld;
          private String Testflag;
          private String Sachbearbeiter;
          private String Telefonnr;
          private String Email;
          private String Berichtszeitraum;
          //Einzelne Meldungen
          private LE1 le1;
          private LE2 le2;
          private LE3 le3;
          private LE4 le4;
          private LE5 le5;
          private LeLimits leLimits;
          //Context
          private Context headerCon;
          private Context NoRCon;
          //C00.01
          //Accounting standard
          private String ei4;
          //Reporting level
          private String ei207;
```

Listing 2: Beispiel des Betragsdatenobjekts

Jede dieser Informationsgruppen hat seinen eigenen Kontext, welcher aus mehreren Informationen besteht. Deshalb wurde hierfür auch eine separate Klasse erstellt, sodass jede Meldung, die Header und der „Nature of Report“ jeweils eigene Kontextobjekte erhalten kann.

Mit diesem Aufbau der Datenobjekte ist die nötige Flexibilität gegeben, um die unterschiedlichen Meldekombinationen abzubilden.

Grob gesagt gibt es zwei Arten, um Daten für die Anwendung anzuliefern. Über die Datenbankschnittstelle und über einen Dateiupload. Bei der Upload-Option muss allerdings noch zwischen den unterschiedlichen Dateiarten differenziert werden. Dabei muss für jeden Dateityp ein Parser zur Verfügung gestellt werden. Um die Anforderung **FS-001** vollständig umsetzten zu können, werden drei Parser benötigt, der Datenbankparser, der CSV-Parser und der Excel-Parser.

Um den Datenbankparser verwenden zu können, muss zuerst die Datenbankverbindung hergestellt und die Tabelle bzw. der View ausgewählt werden. Dazu stehen die in Kapitel 6.2 erläuterten Oberflächen zur Verfügung. Im Hintergrund werden dabei mittels JSF die Daten in Variablen gespeichert. Sind alle relevanten Daten vorhanden, wird eine Verbindung zur

Datenbank aufgebaut. Danach wird das Statement mittels eines *StringBuildes* generiert und ausgeführt.

Abschließend werden die Felder anhand der Feldnamen durch den Parser abgefragt und den jeweiligen Datenobjektfeldern zugeordnet. Dabei wird nach dem Schema, welches in Listing 3 gezeigt wird, vorgegangen.

```
PreparedStatement prepareStatement = conn.prepareStatement(stringBuilder.toString());
ResultSet resultSet = prepareStatement.executeQuery();
while(resultSet.next()){
	Stammdaten stammDaten = new Stammdaten();

	stammDaten.setKreditnehmer(resultSet.getString("Kreditnehmer"));
	stammDaten.setKreditnehmerID(resultSet.getString("KreditnehmerID"));
	stammDaten.setPostleitzahl(resultSet.getString("Postleitzahl"));
```

Listing 3: Ausschnitt des Datenbankparsers

Hierbei wird in einer *while*-Schleife über alle zurückgegebenen Datensätze iteriert und für jeden Datensatz ein neues Objekt angelegt und befüllt. Diese Objekte werden von allen Parsern auf die gleiche Art und Weise befüllt und zur weiteren Verarbeitung genutzt. Damit ist sichergestellt, dass nach dem Parsen eine einheitliche Verarbeitung stattfinden kann.

Im Gegensatz zum Datenbankparser wird der CSV-Parser automatisch getriggert, wenn bei der Uploadmaske, welche in Abbildung 15 dargestellt wird, eine CSV-Datei hochgeladen wird. Das Parsen wird dabei spaltenorientiert durchgeführt. Somit spielt lediglich die Reihenfolge der Datensätze und nicht deren Spaltenname eine Rolle. Dieses Parsen wird in Listing 4 dargestellt und anschließend erläutert.

```
public void parseCSV() {
	String line;
	String csvSplitBy = ";";
	dataList = new ArrayList<String[]>();
	BufferedReader br = new BufferedReader(new InputStreamReader(in));
	while ((line = br.readLine()) != null) {
		dataList.add(line.split(cvsSplitBy));
	}
```

Listing 4: Code Snippet des CSV-Parsers

Bei dem eigentlichen Parsen werden die Daten anhand eines Trennzeichens, welches hier in der Variable *csvSplitBy* gespeichert ist, getrennt. Dabei wird ein *BufferedReader* verwendet, der die zeilenweise Verarbeitung der Daten ermöglicht. Die *while*-Schleife wird dazu benutzt, alle Zeilen durchzuarbeiten. Innerhalb dieser Schleife werden die Zeilen mittels der String-Methode *split* anhand des Trennzeichens getrennt und in einem String-Array gespeichert. Dieses String-Array wird dann der Liste *dataList* hinzugefügt. Mit dieser Liste kann individuell nach Meldung weitergearbeitet werden. Ein Beispiel hierzu wird in Listing 5 dargestellt.

```
public void parseLE1() {
	//Validator Objekt anlegen
	Validator vali = new Validator();
	//Meldungsobjekte anlegen
	LE1 le1 = new LE1();
	Context le1Con = new Context();
	//Prüfung und Füllung der Objekte
	for (String[] data : dataList) {
		if (vali.isNumeric8(data[0])) {
			le1.setCode(data[0]);
		}
```

Listing 5: Beispiel für die Befüllung einer Meldung

Wie in dem Beispiel zu sehen ist, wird zuerst eine Instanz des Validators angelegt. Dieser wird verwendet, um die Daten zu validieren. Danach werden die Objekte für die eigentliche Meldung angelegt. In diesem Fall ein Objekt der LE1 Meldung und deren Kontext. Danach wird die zuvor befüllte Liste *dataList* durchgegangen. Dabei entspricht jede Iteration einem Datensatz und somit einer Meldung. Dadurch wird pro Iteration ein Set der Meldungsobjekte geprüft und befüllt. Als Beispiel hierfür ist das Feld *Code* gezeigt, welches nach der Prüfung in das LE1-Objekt befüllt wird.

Beim Excel-Parser wird die POI-Bibliothek von Apache verwendet. Mit dieser ist es möglich, sowohl das alte Excel-Format XLS sowie das neue Format XLSX zu parsen. Beim Parsen selbst werden die Daten ebenfalls in die *dataList* befüllt, welche in Listing 5 dargestellt ist. Dadurch ist es möglich, die Objektbefüllung des CSV-Parsers zu recyceln.

Nachdem die Datenobjekte befüllt wurden, werden diese entsprechend weiterverarbeitet. Diese Weiterverarbeitung kann den Unterkapiteln 6.5 und 6.7 entnommen werden.

6.4 Datenbank

Wie Abbildung 9, Kapitel 5.1.2 zu entnehmen, ist die interne Datenbank ein wichtiger Bestandteil der Anwendung. In dieser werden neben den Anwendungsdaten, welche dort archiviert werden, auch technische Daten gespeichert. Die Datenbank konnte wie in Abbildung 12 des Kapitels 5.5.1 zu sehen, über die Übersichtsseite der PaaS hinzugefügt werden. Hierbei waren bereits die Datenbanken *MongoDB* und *MySQL* vorinstalliert.[79] Da sich bei diesen Daten das Speichern im relationalen Datenmodell anbietet wurde für die vorliegende Arbeit die Datenbank *MySQL* gewählt. Diese musste lediglich hinzugefügt werden. Zum Verwalten der Datenbank konnte automatisch das Administrationstool *phpMyAdmin* mitinstalliert werden.

[79] Vgl. Red Hat 2016

Die Anwendungstabellen *pdf* und *xbrl*, welche in Abbildung 21 dargestellt sind, können jederzeit exportiert oder ausgelagert werden. Sie dienen lediglich zu Archivierungs- und Nachweiszwecken, welche gesetzlich auf zwei Jahre festgelegt sind.

Sie bestehen aus den Datensätzen der Meldungen und PDF-Dateien. Mittels des *timestamps* und der *userid* können die Datensätze einem bestimmten Bearbeiter und einem bestimmten Meldezeitpunkt zugeordnet werden.
Im Gegensatz dazu beinhalten die technischen Daten PDF-Formulare, XBRL-Taxonomien, benutzerspezifische Daten und Verbindungsdaten, welche zur Verarbeitung benötigt werden. Wie bereits im Unterkapitel 3.4 beschrieben, ist bei einer SaaS-Lösung der Zugriff auf einen lokalen Speicherplatz nicht direkt möglich. Daher wird bei der Speicherung der technischen Daten auf eine Datenbankinstanz ausgewichen. Hier können die Daten jederzeit abgerufen und verarbeitet werden.

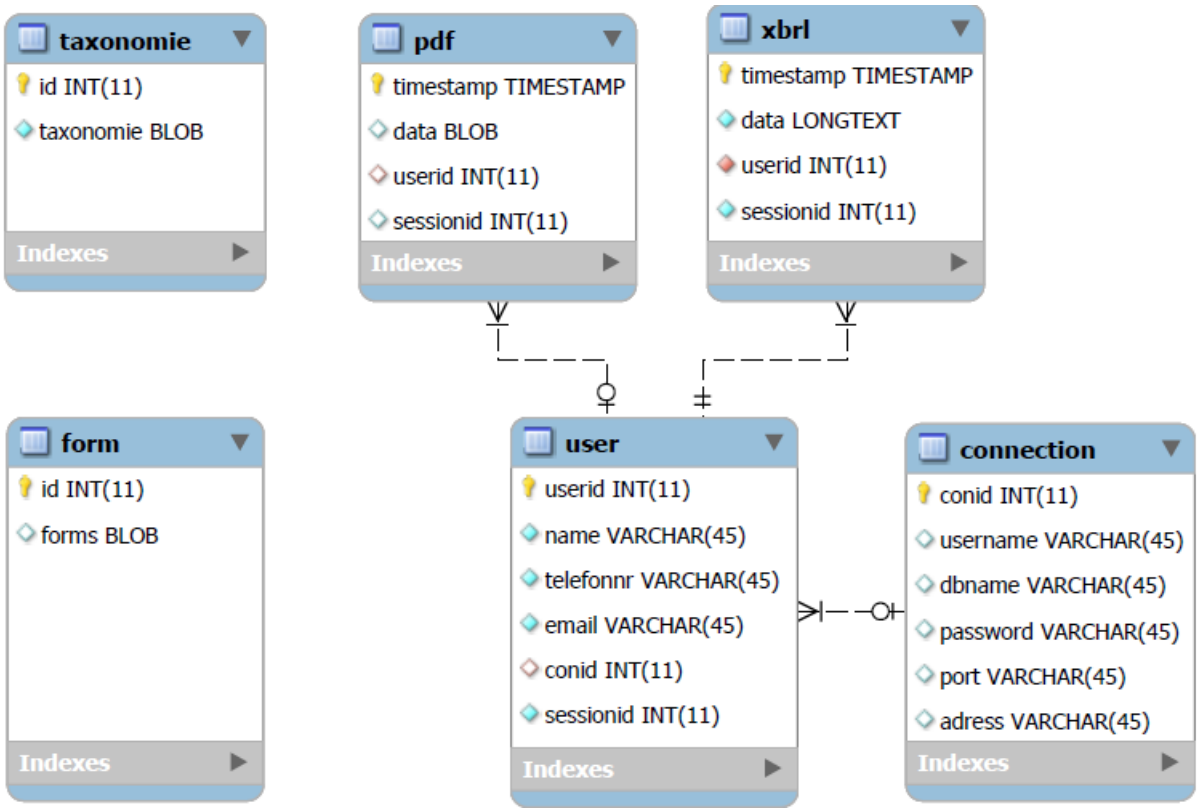

Abbildung 21: Model der internen Datenbank

Konkret werden, wie in Abbildung 21 zu sehen, die PDF-Formulare als Binary Large Objects (BLOB) gespeichert, womit es möglich ist, diese direkt als Stream abzurufen.
Die Taxonomien der Bundesbank sind nicht vollständig online abrufbar. Daher ist die Empfehlung der Bundesbank, die Taxonomien für die Validierungszwecke lokal vorzuhalten.

Dies ist, wie in Kapitel 3.4 beschrieben, bei einer Cloudlösung nur schwer realisierbar. Aus diesem Grund wurden die relevanten Dokumente der Taxonomie ebenfalls als BLOB in die Datenbank eingelesen.
In den Tabellen *user* und *connection* werden individuelle Daten für den Sachbearbeiter gespeichert.

Die benutzerspezifischen Daten bestehen vor allem aus Feldern, die sowohl für die PDF-Formulare wie auch für die Instanz-Dokumente verwendet werden. Diese sind Name, Telefonnummer, Emailadresse und Identifikationsnummern.

Um die Datenbank in Java auslesen zu können, muss zuerst eine Verbindung aufgebaut werden. Dabei wird das Connector-Objekt verwendet. Die Handhabung und das weitere Vorgehen wird in Listing 6 gezeigt.

```
Class.forName(driver);
Connection conn = DriverManager.getConnection("jdbc:mysql://" + adresse + ":" + port +
"/" + dbName, username, password);

PreparedStatement prepareStatement = conn.prepareStatement("SELECT form FROM Formular");
ResultSet resultSet = prepareStatement.executeQuery();
Blob pdfFromular = resultSet.getBlob("form");
```

Listing 6: Aufbau einer Datenbankverbindung und abfragen von Daten

Zu Beginn wird der zu verwendende Treiber spezifiziert. Dazu wird die *forName* Methode mit dem entsprechenden Treiberpfad aufgerufen. Im Falle von MySql wäre es z.B. „com.mysql.jdbc.Driver“. Danach ist mittels dem *DriverManager* die Erstellung eines *Connection*-Objektes möglich, wobei die URL zur Datenbank angegeben werden muss. Nachdem die Verbindung aufgebaut wurde, kann mittels eines Statements eine Objekt Abfrage erstellt werden. In dieser Thesis werden aus Sicherheitsgründen nur *PreparedStatements* verwendet, da diese automatisch Schutz vor SQL-Injektion Attacken bieten.[80] Wie in Listing 6 zu sehen, kann das SQL-Statement einfach der Methode *preparedStatement* des Connection-Objekts als String übergeben werden. Nach der Ausführung des Statements wird ein *ResultSet*, welches die angefragten Daten enthält, zurückgegeben. Mittels der *get* Methoden des *ResultSets* können diese dann abgefragt werden.

Der Vorgang, in eine Datenbank zu schreiben, verläuft, bis zur Ausführung des Statements, analog zum Auslesen. An diesem Punkt wird die Methode *executeUpdate* auf dem

[80] Vgl. Schicker 2014

PreparedStatement ausgeführt. Diese gibt lediglich einen Integer-Wert zurück, der die Anzahl der veränderten Zeilen enthält.

6.5 XBRL-Converter

Der XBRL-Converter übernimmt die Generierung der XBRL-Reports aus den Datenobjekten, wie es in Anforderung **FS-004** gefordert wird. Die Datenobjekte werden zuvor, wie beschrieben, durch die Parser befüllt. Dabei untergliedert sich eine XBRL-Datei in unterschiedliche Bereiche. Diese und deren Generierung werden in den nächsten Abschnitten anhand eines Beispiels erläutert. Dabei wird auf die Grundlagen aus Kapitel 2.3 aufgebaut.

Wie im Listing 7 dargestellt, werden als Erstes die XML-Version und die Codierung definiert. Dies muss bei jedem XML-Dokument angegeben werden. Es folgt ein Root-Element, unter welches sich alle weiteren Elemente gliedern. Dieses enthält die Namespaces und somit Verweise auf benötigte Dokumente und Schemata.

```
<!-- 1. XML-Version und Codierung -->
<?xml version="1.0" encoding="UTF-8"?>

<!-- 2. Root Element mit Namespaces -->
<xbrli:xbrl xmlns:xbrli="http://www.xbrl.org/2003/instance"
        xmlns:de_ext_met="http://www.bundesbank.de/xbrl/dict/met"
        xmlns:eba_met="http://www.eba.europa.eu/xbrl/crr/dict/met"
        xmlns:find="http://www.eurofiling.info/xbrl/ext/filing-indicators"
        xmlns:link="http://www.xbrl.org/2003/linkbase"
        xmlns:xlink="http://www.w3.org/1999/xlink">

        <!-- 3. Definition der Schema Referenz -->
        <link:schemaRef
        xlink:href="http://www.eba.europa.eu/eu/fr/xbrl/corep_le_ind_bbk.xsd"
        xlink:type="simple"/>

        <!-- 4. Definition der Einheiten -->
        <xbrli:unit id="uEur">
                <xbrli:measure>iso4217:EUR</xbrli:measure>
        </xbrli:unit>
        <xbrli:unit id="uPURE">
                <xbrli:measure>xbrli:pure</xbrli:measure>
        </xbrli:unit>
```

Listing 7: Erster Teil eines XBRL-Instanz Dokumentes

Als Drittes wird das Schema referenziert, auf welchem das Instanz-Dokument aufbaut. Dieses Schema dient auch als Anfangspunkt für die Validation. Die drei genannten Schritte sind nur für die technische Verarbeitung nötig.

Als Viertes ist im Listing die Definition der Einheiten dargestellt. In diesem Fall werden die Einheiten „Euro“ und „Prozent“ definiert. Weitere Einheiten werden für das Melden der Großkredite nicht benötigt.

Dieser erste Block des Instanz-Dokumentes wird bei allen Meldungen zum Großkreditmeldewesen sehr ähnlich aussehen, da immer die gleichen Namespaces, Schemata und Einheiten verwendet werden. Daher kann dieser Teil statisch und unabhängig von der eigentlichen Meldung generiert werden.

Ab dem zweiten Teil des Instanz-Dokumentes bekommt jede Elementgruppe einen eigenen Kontext zugewiesen. Dieser besteht, wie in Listing 8 zu sehen, aus einem *context*-Element mit eindeutiger *id*. Unter diesem gliedert sich das Kindelement *entity* mit *identifier*. Dieser enthält die Identifikationsnummer des Kreditnehmers.

```
<xbrli:context id="con02">
	<xbrli:entity>
		<xbrli:identifier
		scheme="http://www.bundesbank.de/identifiertyp/creditorNumber">88888888
		</xbrli:identifier>
	</xbrli:entity>
	<xbrli:period>
		<xbrli:instant>2015-06-30</xbrli:instant>
	</xbrli:period>
	<xbrli:scenario>
		<xbrldi:explicitMemberdimension="eba_dim:BAS">eba_BA:x9
		</xbrldi:explicitMember>
		<xbrldi:explicitMemberdimension="eba_dim:MCY">eba_MC:x59
		</xbrldi:explicitMember>
		<xbrldi:explicitMemberdimension="eba_dim:PRP">eba_PL:x11
		</xbrldi:explicitMember>
		<xbrldi:typedMember dimension="eba_dim:INC">
			<eba_typ:CC>80012891</eba_typ:CC>
		</xbrldi:typedMember>
	</xbrli:scenario>
</xbrli:context>
```

Listing 8: Context Element einer XBRL-Instanz

Des Weiteren ist ein *period* Element enthalten, in welchem, im Fall des Großkredites, nur *instant*-Tags eingefügt werden können, die das Meldedatum enthalten. Der *scenario*-Bereich ist optional. In diesem können die Dimensionen für die Zuordnung einer Elementgruppe zugewiesen werden. Dabei wird zwischen *explicit*- und *typed*-Membern unterschieden. Damit erfolgt beim Auswerten der Daten die Kategorisierung.

Mittels dieser Kontextelemente kann einem Element der Kreditnehmer, das Meldedatum und die Zuordnungsparameter mitgegeben werden. Dabei wird bei dem Element im *contextRef*-Attribut die entsprechende *id* des Kontextes gesetzt.

Die erste Elementgruppe, welche einen solchen Kontext enthält, ist die der *filingIndicators*, siehe Listing 9. Diese geben an, welche Meldungen im Instanz-Dokument enthalten sind. Bei der Großkreditmeldung in Deutschland müssen immer der deutsche Header und der von der EBA definierte „Nature of Report“, welcher als C_00.01 definiert ist, enthalten sein. Diese können somit statisch eingefügt werden. Die anderen Meldungen (LE1-LE5 und LE Limits) müssen, je nachdem ob sie enthalten sind, dynamisch mit den entsprechenden Referenznummern C_26.00 bis C_31.00 hinzugefügt werden. Das Beispiel in Listing 9 enthält den deutschen Header, den „Nature of Report“ der EBA und die Meldungen LE1, LE2 und LE3.

```
<!-- 5. Angabe welche Meldungen enthalten sind -->
<find:fIndicators>
       <find:filingIndicator contextRef="con01-">header</find:filingIndicator>
       <find:filingIndicator contextRef="con01-">C_00.01</find:filingIndicator>
       <find:filingIndicator contextRef="con01-">C_26.00</find:filingIndicator>
       <find:filingIndicator contextRef="con01-">C_27.00</find:filingIndicator>
       <find:filingIndicator contextRef="con01-">C_28.00</find:filingIndicator>
</find:fIndicators>

<!-- 6. Deutscher Header der Bundesbank -->
<de_ext_met:si1 contextRef="con01-">88888888</de_ext_met:si1>
<de_ext_met:si2 contextRef="con01-">creditorNumber</de_ext_met:si2>
<de_ext_met:si3 contextRef="con01-">88888888</de_ext_met:si3>
<de_ext_met:si4 contextRef="con01-">creditorNumber</de_ext_met:si4>
<de_ext_met:si8 contextRef="con01-">test</de_ext_met:si8>
<de_ext_met:si9 contextRef="con01-">Herr Anthony Stark</de_ext_met:si9>
<de_ext_met:si10 contextRef="con01-">555/421337</de_ext_met:si10>
<de_ext_met:si11 contextRef="con01-">tony.stark@TestBank.de</de_ext_met:si11>
<de_ext_met:di12 contextRef="con01-">2015-06-30</de_ext_met:di12>
```

Listing 9: filingIndicatos und Bundesbankheader des Instanz-Dokuments

Als nächstes muss der Header der Bundesbank angegeben werden. Dieser wird in Listing 9 unter 6. dargestellt. In diesem werden wichtige Informationen zum Meldenden, wie Name, Telefonnummer und Email, angegeben. Des Weiteren werden Metadaten, wie Identifikationsnummer und Meldedatum, hinterlegt.

Die meisten Daten für diesen Header können aus der internen Datenbank abgegriffen werden, da dort, wie bereits beschrieben, die Daten zu dem jeweiligen Sachbearbeiter hinterlegt sind. Die restlichen Daten, wie Meldezeitpunkt und Identifikationsnummer, können dynamisch zur Laufzeit befüllt werden.

Nach dem Header folgen die eigentlichen Meldungen mit ihren jeweiligen Kontext-Elementen. Ein Beispiel für eine LE Limits Meldung ist in Listing 10 dargestellt.

```
<!-- LE LIMITS -->
<eba_met:mi49 decimals="-3" unitRef="uEUR" contextRef="con02">87500000.00</eba_met:mi49>
<eba_met:mi47 decimals="-3" unitRef="uEUR" contextRef="con02">150000000.00</eba_met:mi47>
<eba_met:pi48 decimals="4" unitRef="uPURE" contextRef="con02">0.428571</eba_met:pi48>
```

Listing 10: Meldung LE Limits

Wie zu sehen ist, wird bei den Meldungen sowie beim Header mit ID-Tags gearbeitet. Diese haben immer zwei Buchstaben, welche den Typ definieren und eine in ihrem Namespace eindeutige Nummer. Die wichtigsten Typen sind „si“ für String, „pi“ für Prozent, „mi“ für Monetär, „ei“ für Qname und „di“ für Datum. Um diese ID-Tags in menschlich lesbare Terme zu übersetzen benötigt man die passende Label-Linkbase, in welcher Namen zu den jeweiligen IDs zugeordnet werden. Dies hat den Vorteil, dass hier z.B. in unterschiedlichen Ländern verschiedene Label-Linkbases verwendet werden können, aber das eigentliche Instanz-Dokument gleich bleibt.

Die Felder der Meldung orientieren sich an dem dazu passenden Vordruck. In diesem Fall, dem LE Limits Vordruck, welcher in Abbildung 22 abgebildet ist, werden drei Zeilen dargestellt: *Nicht-Institute*, *Institute* und *Institute in %*. Dabei kann direkt erkannt werden, dass die ersten zwei Felder Geldwerte sind und der letzte Wert prozentualer Natur ist. Bei diesen Typen kommen allerdings die gleichen Attribute zum Einsatz. Mittels dieser kann eine zuvor definierte Einheit, die Genauigkeit der Nachkommastellen und ein Kontext zugewiesen werden.

C 26.00 - Obergrenzen für Großkredite (LE Limits)

		Geltende Obergrenze
		Spalte
		010
Zeile		
010	Nicht-Institute	
020	Institute	
030	Institute in %	

Abbildung 22: Vordruck LE Limits[81]

[81] entnommen ausBundesbank 2016b

Nach dem gleichen Prinzip sind auch die anderen Meldungen sowie der „Nature of Report“ aufgebaut. Diese unterscheiden sich lediglich in den ID-Tags und den angegebenen Werten. Die Werte für diese Meldungen werden, wie beschrieben, von den Parsern in die Datenobjekte eingefügt. Diese Datenobjekte werden verwendet, um das Instanz-Dokument zu generieren. Dazu wird, wie in Kapitel 5.3 ausgeführt, der Java XML-Parser benötigt. Hierbei gibt es unterschiedliche Arten, mit XML umzugehen. Für den in dieser Thesis angedachten Zweck ist der Zugriff über Document Object Model (DOM) am besten geeignet, da hier flexibel Elementbäume aufgebaut werden können.[82]

Um mit dem Parser arbeiten zu können muss zuerst ein Document und ein Root Element erzeugt werden. Dieser Vorgang wird in Listing 11 gezeigt.

```
//Vorbereiten des Dokumentbuilders
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
docFactory.setNamespaceAware(true);
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

//Erzeugen und konfigurieren des Documents
doc = docBuilder.newDocument();
doc.setXmlStandalone(true);

// Erzeugen und hinzufügen des Root Elements
Element rootElement = doc.createElement("xbrli:xbrl");
doc.appendChild(rootElement);
```

Listing 11: Erzeugung eines XBRL-Dokuments

Zu Beginn kommt eine *DocumentBuilderFactory* zum Einsatz. Bei jenem muss das Arbeiten mit Namespaces aktiviert werden. Dies ist, wie bereits beschrieben, ein wichtiger Bestandteil von XBRL-Instanzen. Danach kann ein *DocumentBuilder* erstellt werden, welcher in Folge ein *Document* instanziiert. Hierbei muss aufgrund von Vorgaben der Bundesbank, das Standalone-Flag auf *true* gesetzt werden. Nach diesen Schritten ist das Document-Objekt fertig initialisiert und kann befüllt werden. Dabei beginnt jedes XML-Dokument mit einem Wurzelelement, das auch Root-Element genannt wird. An dieses Element werden dann alle weiteren Elemente als Kindknoten angehängt. Um ein Element zu erzeugen, wird die Methode *createElement* auf dem Document-Objekt aufgerufen. Als Übergabeparameter muss hierbei der *Tag* des zu erzeugenden Elements übergeben werden. Nach der Erstellung des Wurzelknotens muss dieser an das *Document* angefügt werden. Dies wird mit der Methode *appendChild* realisiert.

[82] Vgl. 2010 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT 2010)

Nachdem das Dokument und das Root-Element erstellt und hinzugefügt wurden, können die Meldungsdaten angefügt werden. Dabei sind vor allem die Methoden in Listing 12 von Relevanz.

```
// Erzeugen und Hinzufügen des Sachbearbeiters
Element headerElement = doc.createElement("de_ext_met:si9");
headerElement.setAttribute("contextRef", "con01");
headerElement.appendChild(doc.createTextNode("Bruce Wayne"));

rootElement.appendChild(headerElement);
```

Listing 12: DOM-Methoden am Beispiel des Sachbearbeiters

In diesem Beispiel wird die Erstellung und das Hinzufügen eines Knotens gezeigt. Dabei wird die bereits erwähnte Methode *createElement* des *Documents* verwendet, um den Knoten zu erstellen. Auf dem erstellten Element können danach noch Modifikationen, wie z.B. das Hinzufügen von Attributen mittels der Methode *setAttribute*, durchgeführt werden. Des Weiteren kann durch das Anfügen eines Textknotens der eigentliche Wert gesetzt werden. Der Textknoten kann mittels *creatTextNode* von dem *Document*-Objekt erzeugt werden. Als Übergabeparameter muss dabei der Wert übergeben werden, welchen der Textknoten annehmen soll. Nach Erzeugung des Textknotens wird dieser durch die Methode *appendChild* zu dem Knoten hinzugefügt, welcher die Methode aufgerufen hat. In diesem Fall ist das der Knoten *headerElement*. Wurden alle Werte und Modifikationen am Knoten vorgenommen, kann dieser an den Elternknoten angefügt werden. In diesem Beispiel wird der Knoten direkt an den Wurzelknoten angefügt, was wiederum mit der *appendChild*-Methode erflogt.

Mit diesen Methoden ist es möglich, alle Elemente, die zur Erstellung des Instanz-Dokumentes nötig sind, zu erzeugen. Sind alle Elemente erzeugt und zusammengefügt, werden die fertigen XBRL-Daten mit dem bereits beschriebenen XBRL-Validator validiert. Ist der Validierungsvorgang erfolgreich abgeschlossen, werden die Instanz-Dokumente in einen String konvertiert. Anschließend werden diese mit einem Zeitstempel, der User-ID und der Session-ID in die Datenbank geschrieben. Somit kann jederzeit nachgewiesen werden, welcher Benutzer zu welchem Zeitpinkt eine Meldung erzeugt hat. Die Validierung und Umwandlung ist in Listing 13 zu sehen.

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
DOMSource source = new DOMSource(doc);

if (XbrlValidator.validate(source)) {

	StringWriter outWriter = new StringWriter();
	StreamResult result = new StreamResult(outWriter);
	transformer.transform(source, result);

	StringBuffer sb = outWriter.getBuffer();
	String finalstring = sb.toString();

	writeInDb(finalstring);
}
```

Listing 13: Umwandeln des generierten Instanz-Dokuments

Die Validation wird mittels des bereits erwähnten XBRL-Validators durchgeführt. Dabei wird das fertige Instanz-Dokument an diesen übergeben. Für die Umwandlung wird ein Transformer benötigt. Dieser kann mittels der *transform*-Methode ein Source-Objekt in ein Result-Objekt umwandeln. Wie im oben dargestellten Listing zu sehen wird hierbei *DOMSource* in *StreamResult* umgewandelt. Danach kann mittels der Methode *getBuffer* ein *StringBuffer* aus dem im *result* enthaltenen *StringWriter* extrahiert werden. Vom *StringBuffer* kann anschließend mit *toString* ein String erhalten werden.
Mit der Methode *writeInDb* wird der erzeugte String in die Datenbank geschrieben. Die Methode ist analog zu der in Kapitel 6.4 beschriebenen aufgebaut.

Der gespeicherte XBRL-String wird abschließend im XBRL-Viewer, welcher in Abbildung 20 dargestellt ist, zur Kontrolle angezeigt.
Um zusätzlich den Download des Instanz-Dokumentes zu erlauben, muss der String in ein *StreamedContent*-Objekt umgewandelt werden. Dazu wird der String zuerst in einen *InputStream* umgewandelt, welcher dann als Inputparameter für den Konstruktor des *DefaultStreamedContent*-Objekts verwendet werden kann.

6.6 Validation

Die Validation von Daten ist ein sehr wichtiger Bestandteil dieser Thesis, da nur damit die Richtigkeit der Daten sichergestellt werden kann. Nicht-Valide Daten werden vom Extranet nicht angenommen, sondern direkt abgewiesen. Die Validation unterteilt sich hierbei in mehrere Schichten. Diese sind in Abbildung 23 dargestellt und werden im folgenden Abschnitt inklusive der Umsetzung erläutert.

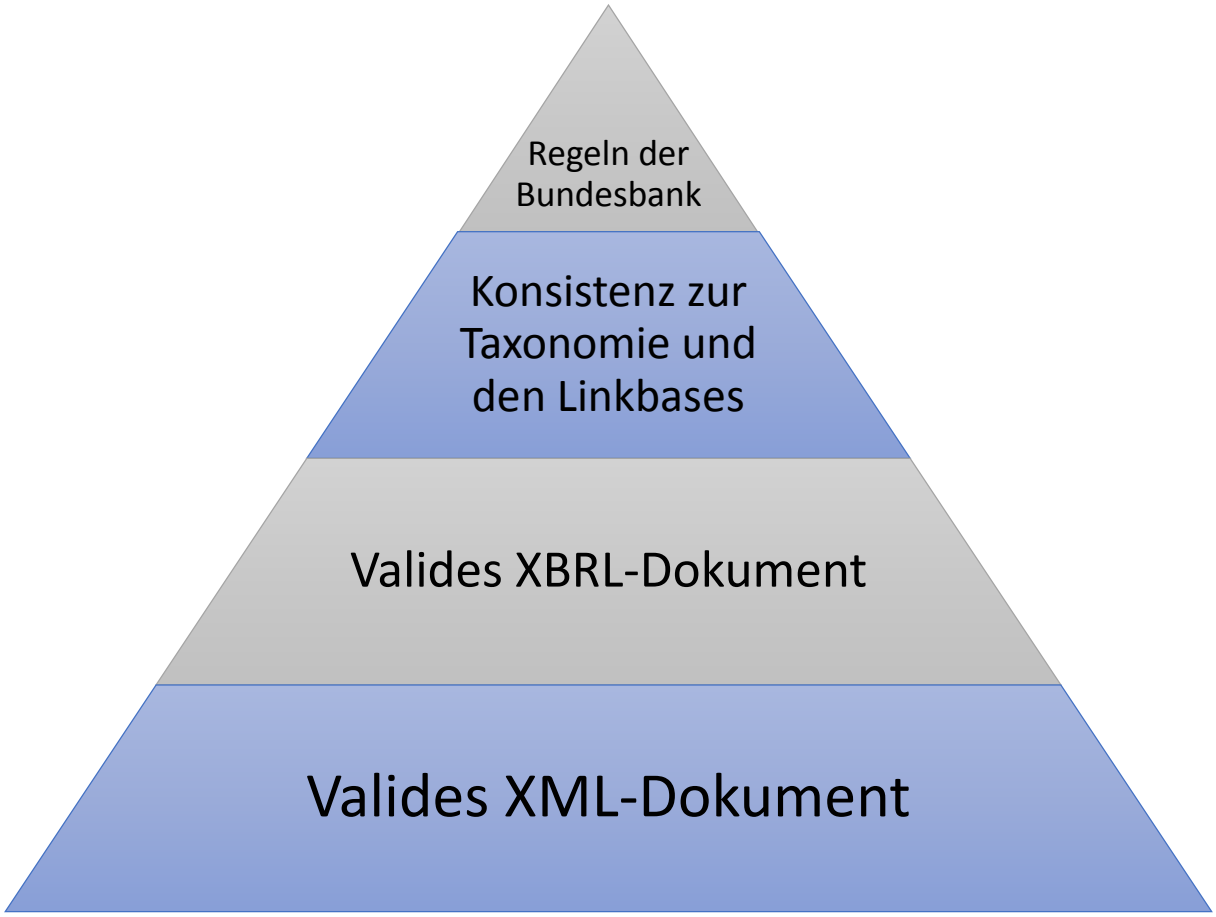

Abbildung 23: Validationsebenen von XBRL-Dokumenten

Bei der Abbildung bildet die unterste Schicht die Grundlage; es wird nach oben hin immer feiner granuliert. Dabei können die unteren drei Punkte erst nach Erzeugung des Instanz-Dokumentes validiert werden. Das liegt daran, dass erst dann die Struktur der Daten festliegt und diese mit den Schemata und Taxonomien verglichen werden kann. Der oberste Punkt bezieht sich nur auf die Daten und kann daher bereits beim Einlesen geprüft werden. Somit entsteht ein zweigeteilter Validierungsprozess. Im ersten Unterkapitel wird dabei der Regel-Validator erläutert, welcher den oberen Teil der Pyramide übernimmt. Im zweiten Teil wird die Struktur der Datei validiert.

6.6.1 Regel-Validator

Der Regel-Validator wird eingesetzt, bevor mit der Verarbeitung der Daten begonnen wird. Dabei werden die Daten, wie in Anforderung **FS-002** festgelegt, auf Plausibilität geprüft. Dies wird anhand der Bundesbankvorgaben, welche in einer Plausibilitätsliste beschrieben sind, getätigt. Geprüft wird dabei die Richtigkeit des Datentyps des Feldes und zum Teil auch die Anzahl der zulässigen Zeichen. Die Beschreibung der kompletten Validation würde an dieser

Stelle zu weit führen und ist für das Verständnis der Thesis unerheblich. Daher wird die konkrete Prüfung im Folgenden anhand eines Beispiels veranschaulicht.

Zuerst werden die Anforderungen in der Plausibilitätsliste, von welcher ein Auszug in Abbildung 24 zu sehen ist, geprüft. In dieser dreigeteilten Tabelle kann im vorderen Teil abgelesen werden, welche Meldungen betroffen sind. Im mittleren Teil erfolgt eine Beschreibung der Validation und im dritten Teil werden die betroffenen Felder beschrieben.

Großkredit-Plausibilitätsprüfungen*

gültig ab Meldestichtag 31.03.2016

betroffene Vordrucke							Beschreibung der Plausibilität **	Großkredit-Position/en relevant für Plausibilitätsprüfung
allge-mein	LE 1	LE 2	LE 3	LE 4	LE 5	LE-Li-mits		
x	x	x	x	x	x		Identifikationsnummer Format achtstellig, nummerisch	Melder in Basis-Taxonomie, LE1.010, LE2.010, LE3.010, LE3.020, LE4.010, LE5.010, LE5.020
x	x	x	x	x	x		gültige Prüfziffer, Prüfziffernrechung s. Merkblatt für die Abgabe der Groß- und Millionenkreditanzeigen Teil VI Anlage 3	Melder in Basis-Taxonomie, LE1.010, LE2.010, LE3.010, LE3.020, LE4.010, LE5.010, LE5.020
x	x	x	x	x	x		Melder und Kreditnehmer dürfen nicht identisch sein	Melder in Basis-Taxonomie, LE1.010, LE2.010, LE3.010, LE4.010, LE5.010
x							Zusatzangaben Format maximal 20-stellig, alphanummerisch	Zusatzangaben in Basis-Taxonomie
	x	x	x	x	x		"Code" ist Pflichtfeld	LE1.010, LE2.010, LE3.010, LE4.010, LE5.010
	x						KN/GvK-Nummern in LE1 müssen eindeutig sein	LE1.010
	x	x	x				wenn LE1 vorhanden, dann muss LE2 oder LE3 vorhanden sein	LE1.010, LE2.010, LE3.010, LE3.020
	x						LEI Format alphanummerisch, genau 20-stellig	LE1.030

Abbildung 24: Auszug aus der Plausibilitätsliste für Großkredite der Bundesbank[83]

Wie in der ersten Zeile zu sehen ist, muss die Identifikationsnummer in allen Meldungen außer der LE Limits enthalten sein. Das Format wird dabei als achtstellig und nummerisch festgelegt. Der Ausschnitt der Tabelle zeigt, dass Regeln meist nicht nur für eine Meldung definiert werden. Dies macht es zusätzlich zu der fachlichen Komponententrennung sehr sinnvoll, die Validation in eine eigene Komponente auszulagern, da somit die Methoden recycelt werden können.

[83] siehe Bundesbank 2016a

Um die genannte Anforderung in Java umzusetzen, wird mit Regular Expressions (RegEx) gearbeitet. Mit diesen ist es möglich, den Datentyp, die Länge und sonstige Formatangaben von Feldern zu überprüfen.[84] In Listing 14 wird die Prüfung der Identifikationsnummer gezeigt.

```
public class Validator {

    public boolean isNumeric8(String value) {
        return value.matches("\\d{8}");
    }
```

Listing 14: isNumeric Methode aus der Validator Klasse

Dabei wird der String an die Methode übergeben. Mittels der *matches* Methode kann ein RegEx Ausdruck angegeben werden. Mit dem Parameter *d* wird angegeben, dass der String nur Zahlen enthalten darf. Die darauffolgenden geschweiften Klammern dienen dazu, die Länge festzulegen. In diesem Fall muss es also, wie gefordert, eine achtstellige Zahl sein. Wenn der übergebene String den Anforderungen entspricht, wird *true* zurückgegeben. Wenn er den Anforderungen hingegen nicht entspricht, wird *false* zurückgegeben. Somit kann diese Methode sehr gut zur Prüfung eingesetzt werden, da sie beim Parsen in einem If-Statement verwendet werden kann.

Mit RegEx können allerdings auch weitaus komplexere Formate abgeprüft werden, wie z.B. ein Datum, welches mit Bindestrichen getrennt sein muss.[85] Somit ist sichergestellt, dass alle Validationen mit diesem Verfahren durchgeführt werden können.

Die anderen Punkte der Plausibilitätsliste wurden nach demselben Prinzip in der Validator-Klasse angelegt und entsprechend beim Parser aufgerufen.

6.6.2 XBRL-Validator

Der XBRL-Validator beschäftigt sich mit den unteren drei Punkten der in Abbildung 23 dargestellten Pyramide. Diese umfassen die XML-Validation, die XBRL-Validation und die Konsistenz der Daten zu Taxonomien und Linkbases.

Die XML- und XBRL-Validation kann normalerweise über den Standard Java-XML-Validator realisiert werden. Im Rahmen der vorliegenden Bachelor-Thesis war dies allerdings nicht möglich, da die Validationskette der Schemata unterbrochen wurde. In diesem konkreten Fall bedeutet das, dass in den Schemadateien ungültige Verlinkungen zu weiteren Schemadateien enthalten sind. Auf Anfrage bei der Bundesbank zu diesem Problem wurde ausgesagt, dass die

[84] Vgl. 2010 proceedings, IEEE INFOCOM 2010
[85] Vgl. 2010 proceedings, IEEE INFOCOM 2010

EBA nicht alle Schemadateien der Bundesbank unter den angegebenen Links bereitstellt. Die Empfehlung der Bundesbank war für diesen Fall, die Taxonomien lokal vorzuhalten. Dies konnte zwar, innerhalb der Cloudumgebung, mittels der Datenbank gelöst werden, der geplante Einsatz des Standard XML-Validators war dadurch allerdings nichtmehr möglich.

Unter diesen Umständen konnte die Validation aus zeitlichen Gründen nicht umgesetzt werden. Lediglich eine Prüfung, ob die Struktur wohlgeformt ist, konnte durchgeführt werden. Für die produktive Umsetzung dieses POC müsste allerdings die Validation um die entsprechenden Schritte erweitert werden, um sicherzustellen, dass valide Daten an die Bundesbank gesendet werden.

6.7 PDF-Builder

In diesem Unterkapitel wird, wie in Anforderung **FS-003** definiert, der Aufbau des PDF-Builders bzw. der Build-Methode beschrieben. Hierbei wird, die in Kapitel 5.4 gewählte, PdfBox-Bibliothek von Apache eingesetzt, um ein PDF-Formular zu befüllen. Die zugehörige Klasse wird in dem folgenden Listing vereinfacht gezeigt und anschließend beschrieben. Dabei werden Error Handling, Logging und Zeilen mit ähnlicher Funktion aus Übersichtlichkeitsgründen nicht mit einbezogen.

```
public StreamedContent build(Stammdaten sta)
{

        //get pdfobject
        PDDocument tempPdf = pdfTemplate;
        PDDocumentCatalog docCatalog = tempPdf.getDocumentCatalog();
        PDAcroForm acroForm = docCatalog.getAcroForm();

        //Set Data
        acroForm.getField(PLZ).setValue(sta.getPostleitzahl());
        acroForm.getField(SITZ).setValue(sta.getSitz());
        acroForm.getField(ISO).setValue(sta.getISO());
        acroForm.getField(BERUF).setValue(sta.getBeruf());
        acroForm.getField(GEBURTSDATUM).setValue(sta.getGeburtsdatum());
        acroForm.getField(KREDITNEHMER1).setValue(sta.getKreditnehmer());
        acroForm.getField(KREDITNEHMER_ID).setValue(sta.getKreditnehmerID());
        acroForm.getField(K_EINZELINSTITUT).setValue("true");

        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        tempPdf.save(baos);

        InputStream finalPdf = new ByteArrayInputStream(baos.toByteArray());

        StreamedContent sc = new DefaultStreamedContent(finalPdf,
        "application/pdf",sta.getTabName());
        return sc;
}
```

Listing 15: Code des PDF-Builders

Wie in Listing 15 zu sehen, wird der *build*-Methode ein Objekt der Stammdaten übergeben. Dort sind die relevanten Daten zur Befüllung des Formulars enthalten. Prinzipiell muss zuerst das Formular geladen werden. Dies wird mit den ersten drei Befehlen umgesetzt. Dabei enthält die Variable *pdfTemplate* bereits das Formular. Dieses wird einmalig aus der Datenbank geladen und in dieser Variable hinterlegt. Danach wird über den *Catalog* die *AcroForm* aus dem PDF extrahiert. Diese ist das eigentliche Formular, welche die zu bestückenden Felder enthält.

Über die *getField* Methode und den entsprechenden Feldnamen kann nun ein Feld angesprochen werden. Die Feldnamen wurden in einem vorbereitenden Schritt bereits ausgelesen und in Konstanten mit sprechenden Namen gespeichert. Wenn das entsprechende Feld angesprochen wurde, kann über *setValue* der jeweilige Wert gesetzt werden. Dieser Wert wird, wie in vorhergehendem Listing zu sehen, aus dem übergebenen Stammdatenobjekt über die getter-Methoden geholt.

Sind alle Felder befüllt, muss das PDF in einen *StreamedContent* umgewandelt werden, um dieses im PDF-Viewer anzeigen zu können. Da es hierfür keinen direkten Weg gibt, muss das PDF erst als *OutputStream* gespeichert werden, welcher anschließend in einen *InputStream* umgewandelt wird. Mit diesem ist es letztendlich möglich, ein *StreamedContent* zu erzeugen. Bei dessen Erzeugung ist es möglich, den Content-Typ und den Namen anzugeben. Der Content-Typ wird dabei statisch als „pdf“ gesetzt; der Name wird dynamisch aus dem Stammdaten-Objekt geholt.

Der erzeugte *StreamedContent* wird anschließend als BLOB in der Datenbank gespeichert. Als Metainformationen werden zusätzlich ein Zeitstempel, die User-ID und die Session-ID gespeichert. Somit ist es jederzeit möglich zurückzuverfolgen, wann welcher User eine bestimmte Meldung erstellt hat. Abschließend werden die gespeicherten PDF-Dateien in dem PDF-Viewer, welcher in Abbildung 19 dargestellt ist, angezeigt. Mit diesem ist es möglich, das PDF herunterzuladen oder zu drucken.

6.8 ExtraNet Schnittstelle

Um die Daten an die Bundesbank übermitteln zu können, wird eine Schnittstelle zum System der Bundesbank, des sogenannten ExtraNet, benötigt. Im Rahmen dieser Thesis konnte von der Bundesbank kein Testzugang eingerichtet werden. Daher konnte die Anforderung **FS-005** und somit die Implementierung nicht umgesetzt werden. Die Dokumentation beschreibt jedoch sehr

genau, wie die Kommunikation mit dem ExtraNet ablaufen muss. Aufgrund dieser Beschreibung konnte der Vorgang zum Senden bzw. Uploaden der Daten sowie die Erhaltung und Verarbeitung einer Rückantwort extrahiert werden.

Mit dem ExtraNet kann nur eine über TLS 1.2 verschlüsselte Verbindung aufgebaut werden. Zusätzlich ist die Verbindung mittels eines Zertifikats gesichert. Dieses muss im Voraus importiert werden, da ansonsten das Zertifikat als unsicher abgewiesen werden könnte. Des Weiteren wurde darauf hingewiesen, dass bei jedem Request alle in der Response gesendeten Cookies mit zurückgeschickt werden. Andernfalls kann der reibungslose Ablauf nicht gewährleistet werden.
Sind diese Voraussetzungen erfüllt, kann das Login mit dem Ablauf, welcher in Abbildung 25 gezeigt wird, durchgeführt werden.

Stufe	Beschreibung
1	Client → Server : GET protocol://hostname/your-contents-name
2	Client ← Server : FORM wird geschickt mit Session-Cookie (PD-S-SESSION-ID bei HTTPs)
3	Client → Server : POST protocol://hostname/pkmslogin.form mit Session-Cookie und userid&password&login-form-type
4	Client ← Server : 302 redirect zur angeforderten Seite
5	Client → Server : GET protocol://hostname/your-contents-name mit Session-Cookie
6	Client ← Server : Angeforderte Seite wird geschickt

Abbildung 25: Loginprozess des ExtraNets[86]

Zuerst wird das Login Formular angefordert (1). Nachdem dieses zurückgesendet wurde (2), können die Parameter *userid* und *password* an den Server gesendet werden (3). Nach deren Prüfung wird man auf die ExtraNet-Startseite weitergeleitet (4). Anschließend können die entsprechenden Servlets des ExtraNets genutzt werden (5,6). Für das Uploaden von Dateien steht ein spezielles Upload-Servlet bereit, welches den Dateityp und den Dateninhalt in einem POST-Request erwartet.

[86] entnommen ausBundesbank 2015b

Nachdem die Datei hochgeladen wurde, wird eine XML Seite, wie die in Listing 16 dargestellte, zur Überprüfung zurück gesendet. Diese wird wiederum mittels eines XML-Parsers ausgewertet und auf Richtigkeit überprüft. Dabei kann z.B. der Dateiname und die Dateigröße entsprechend gegengeprüft werden.

```
<upload>
	<dateiname>Betragsdaten.xbrl</dateiname>
	<dateigroesse>3</dateigroesse>
	<einstellzeit>18.01.16/14:59</einstellzeit>
	<auftragskennung>
		<sender>FTSEX</sender>
		<empfaenger>NACHS</empfaenger>
		<typ>SD</typ>
		<auftragsid>1065</auftragsid>
	</auftragskennung>
</upload>
```

Listing 16: Upload-Bestätigung des ExtraNets

6.9 Testen der RaaS-Lösung

Um die Funktionalität der RaaS-Lösung und somit die Umsetzung der Anforderung **NFS-001** sicherzustellen, wurden verschiedene Tests durchgeführt. Für diese Zwecke wurde die in Kapitel 5.5 beschriebene Testumgebung verwendet. Dort konnte die Anwendung in einer PaaS mit Load-Balancer ausgeführt werden, was den Einsatzbedingungen entspricht.
Da allerdings weder eine Serverfarm, noch ein Clientnetz zur Verfügung stand, konnte das Load-Balancing und die Skalierung nur in sehr geringem Maße überprüft werden.

Für das Testen wurden Testdaten in Datenbanktabellen und Dateien vorbereitet. Dabei wurde darauf geachtet, sowohl den positiven wie auch den negativen Fall abzudecken. Die erstellten Testdaten sind in Tabelle 3 beschrieben. Die Tests selbst werden in Tabelle 5 aufgelistet , wobei die Testdaten, welche für den jeweiligen Test verwendet wurden, referenziert werden. Die Präfixe der Nummern geben dabei schon an, ob es sich um Stamm- oder Betragsdaten handelt. Der Test an sich wird in der Spalte „Beschreibung“ dokumentiert. Die Spalte „Ergebnis“ gibt Auskunft über das gelieferte Ergebnis.

Testnummer	Beschreibung	Testdaten	Ergebnis

TB-001	Alle Datensätze müssen ohne Probleme verarbeitet werden.	BD-001	Erfolgreich
TB-002	Die Datensätze sind vollständig, die Werte sind unzulässig, der Parser muss die Daten mit der Fehlermeldung „nicht valide Daten“ ablehnen.	BD-002	Daten abgelehnt
TB-003	Die Datensätze sind unvollständig, der Parser muss die Daten mit der Fehlermeldung „unvollständige Daten“ ablehnen.	BD-003	Daten abgelehnt
TS-001	Alle Datensätze müssen ohne Probleme verarbeitet werden.	SD-001	Erfolgreich
TS-002	Die Datensätze sind vollständig, die Werte sind unzulässig, der Parser muss die Daten mit der Fehlermeldung „nicht valide Daten“ ablehnen.	SD-002	Daten abgelehnt
TS-003	Die Datensätze sind unvollständig, der Parser muss die Daten mit der Fehlermeldung „unvollständige Daten“ ablehnen.	SD-003	Daten abgelehnt

Tabelle 5: Beschreibung der Tests

Zusätzlich zu diesen Tests wurden die leeren Dateien und Views sowie unzulässige Dateiformate getestet. Des Weiteren werden, wie bereits beschrieben, die Eingaben des Users auf Plausibilität überprüft und durch Elemente wie Dropdowns eingeschränkt. Dadurch entsteht grundsätzlich ein wesentlich geringeres Fehlerpotenzial.

Während der Entwicklung der Anwendung wurde die Qualität mittels Unit-Tests gesichert. Da es sich allerdings nur um einen POC handelt, wurde kein Testabdeckungsmaß festgelegt. Der Schwerpunkt wurde hierbei auf die grundsätzliche Funktionalität und Machbarkeit gelegt. Diese wurden mit den oben genannten Tests schon völlig abgedeckt und erfüllt.

7 Kritische Reflexion

In diesem Kapitel wird die Thesis kritisch reflektiert. Dabei wird vor allem auf Schwierigkeiten und Probleme eingegangen, welche bei der Umsetzung einer RaaS-Lösung im Meldewesen auftreten können. Dies schließt sowohl wirtschaftliche wie auch technische Bedenken ein.

Bei dem Vergleich der bestehenden Lösungen war klar zu sehen, dass gut positionierte Meldesoftware-Unternehmen wie Abacus bereits erste Schritte unternommen haben, um ihre Anwendungen in die Cloud zu portieren. Diese Unternehmen verfügen nicht nur über die technische Expertise, sondern auch über die entsprechenden Marktanteile und Referenzen, um sehr schnell umfangreiche Cloud-Lösung bereitstellen zu können.
Des Weiteren drängen Unternehmen, welche bereits XBRL-Cloudlösungen anbieten in Richtung Meldewesen. Daher sollte der Markt vor dem Einstieg mit einer eigenen Software genauer betrachtet werden.
Darüber hinaus wäre eine wirtschaftliche Abwägung, ob die Skalierung und somit die Umsetzung in einer Cloud wirklich nötig sind, sinnvoll. Als Alternative könnte für kleine Unternehmen ein Web-Service eingesetzt werden. Dieser erfordert kein großes Rechenzentrum und kann auf einem einzelnen Server gehostet werden. Dabei sollte die Last, mit welcher die Anwendung frequentiert wird, geprüft werden.

Auf technischer Seite ist vor allem die schlechte Verfügbarkeit von kostenlosen XBRL-Tools und Bibliotheken ein Problem. Auch die Bundesbank stellt hier keinerlei Tools zur Verfügung. Daher müssen entweder kommerzielle Tools gekauft oder eine aufwendige Eigenentwicklung umgesetzt werden.
Die Erzeugung der XBRL-Dateien und das Validieren ebendieser wurde in dieser Thesis selbst implementiert. Dabei wurden bei der Validierung allerdings nicht alle nötigen Schritte umgesetzt, um eine fehlerfreie Datei zu garantieren. Für eine Umsetzung aller Validierungsmechanismen muss wesentlich mehr Zeit einkalkuliert werden. Die Validierung wurde, wie beschrieben, zusätzlich dadurch verzögert, dass die Bundesbank nicht alle Schemata bei der EBA hinterlegt hat und diese dadurch lokal vorgehalten werden müssen. Dadurch war es nicht möglich, die Anforderung **FS-004** in vollem Maße umzusetzen.
Die Implementierung von SSO wurde bewusst nicht durchgeführt, da die SSO-Landschaft in den Unternehmen zu heterogen ist und somit keine Universallösung möglich ist. Das System erfüllt allerdings die Anforderungen für die Anbindung an ein bestehendes System.

8 Fazit und Ausblick

In diesem abschließenden Kapitel wird eine Zusammenfassung in Form eines Fazits gezogen. Anschließend wird ein Ausblick in Bezug auf die Möglichkeiten des vorgestellten Lösungsansatzes gegeben. Des Weiteren werden die Veränderungen des Meldeumfelds und die dadurch entstehenden Möglichkeiten durchleuchtet.

8.1 Fazit

Aufgrund der stetig wachsenden Anforderungen im Meldewesen müssen die Meldungen in Zukunft maschinell durchgeführt werden. Dabei hat sich XBRL als Standard durchgesetzt, da er nicht nur von der EBA, sondern auch von amerikanischen und indischen Behörden eingesetzt wird.

Die Meldesysteme müssen eine gewisse Flexibilität in der Umsetzung bieten. Cloud Systeme sind optimal für diesen Zweck geeignet, da der Zugang zur Wartung und Weiterentwicklung nicht auf den Endsystemen, sondern zentralisiert stattfindet. Auch der Kostenfaktor spielt dabei eine große Rolle. So können Cloud Systeme auch kleinen und mittelständischen Finanzunternehmen einen Zugang zu Meldesystemen ermöglichen.

Auch der bestehende Markt zeigt, dass bereits eine Tendenz Richtung Cloudlösungen besteht. So streben nicht nur cloudbasierte XBRL-Tools Richtung Meldewesen, sondern auch desktopbasierte Meldesysteme Richtung Cloud.

Ein Problem zeichnet sich in der freien Verfügbarkeit von XBRL-Prozessoren ab, welche dringend benötigt werden, um effizient mit XBRL arbeiten zu können. Vor allem die Validierung ist in diesem Fall der Knackpunkt, da diese bei XBRL sehr umfangreich ist und dadurch eine komplexe Implementierung mit sich bringt. So muss bei der Entwicklung von XBRL-Anwendungen auf kommerzielle Tools zurückgegriffen oder eine aufwendige Eigenentwicklung durchgeführt werden.

Mit der, in dieser Thesis entwickelten, Anwendung kann ein kompletter Durchlauf simuliert werden. Dieser geht von der Anmeldung über das Einlesen von Daten bis hin zur fertigen Meldung, die dann entsprechend gedruckt oder gesendet werden kann. Dabei wurden aus zeitlichen oder organisatorischen Gründen einige Punkte nicht vollständig umgesetzt oder nur theoretisch beschrieben. Dazu zählen die Umsetzung von SSO und die vollständige Validierung der XBRL Dateien. Des Weiteren konnte von der Bundesbank kein Testzugang für das ExtraNet bereitgestellt werden, weshalb diese Schnittstelle ebenfalls nur beschrieben wurde.

Durch Umsetzung als SaaS zeigten sich bereits bei der Entwicklung Vorteile. Die Testclients konnten immer direkt auf die neue Version zugreifen, sobald diese in der Cloud bereitgestellt wurde. Des Weiteren war der parallele Zugriff von mehreren Clients durch die automatisch erzeugten Instanzen kein Problem.
Die Testumgebung ist im Vergleich zu den realen Einsatzbedingungen unterdimensioniert, weshalb keine Last- oder Skalierungstests durchgeführt werden konnten. Die Vorteile der Skalierung und Lastverteilung werden sich erst in der Produktivumgebung zeigen. Ähnlich verhält es sich mit den Kostensenkungen, welche erst beim Vergleich mit existierenden Systemen und dem Einsatz in den Firmen berechnet werden können.

Zusammengefasst zeigt die Thesis, dass die Entwicklung eines XBRL basierten RaaS für das Meldewesen gut umsetzbar ist und auch in vielen Gebieten Einsatz findet. Des Weiteren deckt die Thesis die Schwerpunkte auf, die bei der Entwicklung einer solchen Anwendung beachtet werden müssen. Im folgenden Unterkapitel werden diese erläutert und Empfehlungen zur Weiterentwicklung ausgesprochen.

8.2 Ausblick

Der POC, welcher in dieser Thesis erstellt wurde, kann sehr gut als Grundlage für die tatsächliche Umsetzung herangezogen werden. Darauf aufbauend können mehrere Meldungen und Taxonomien unterstützt werden. Auch der Input und Output kann um weitere Möglichkeiten erweitert werden. So könnten noch weitere Datei-Formate oder Datenbanktreiber unterstützt werden. Mit den Daten ließen sich Reports erstellt, welche z.B. vom Rechnungswesen oder anderen Abteilungen genutzt werden können. Ein darauf aufbauendes Auswertungssystem, welches ein Risikomanagement mit unterschiedlichen Warnstufen enthält, wäre auch denkbar.

Auch eine Verbesserung der bestehenden Schnittstellen wäre vorstellbar. Es könnte z.B. ein Verknüpfungsmanager bereitgestellt werden, mit welchem es möglich ist, Felder eines Views mit den entsprechenden Werten in der XBRL oder PDF Datei zu verknüpfen. Dieser birgt jedoch die Gefahr, dass er bei zu vielen Feldern unübersichtlich wird.

In Zukunft werden die Stammdaten ebenfalls elektronisch übertragen. In diesem Fall wäre eine Umstellung der Stammdatenmeldung von PDF auf XBRL sinnvoll.

Die Validatoren müssen verbessert und erweitert werden, da sonst die Daten eventuell vom ExtraNet nicht akzeptiert werden. Dies würde auch eine qualitative Verbesserung der Daten

nach sich ziehen. Des Weiteren ist ein Validationsmanager, bei dem neue Regeln eingetragen oder bestehende Regeln gepflegt werden, eine sehr sinnvolle Erweiterung. Damit kann eine bessere Trennung des fachlichen und des technischen Bereiches der Anwendung umgesetzt werden.

Da sich der XBRL-Standard nicht nur im Meldewesen, sondern auch in anderen Bereichen und Branchen ausbreitet, ist das Erweiterungspotenzial der Anwendung sehr hoch. Dies sollte nach der Prüfung der in Kapitel 7 genannten Risiken auch ausgeschöpft werden. Wenn eine solide Basis der Anwendung geschaffen wurde, stellt die Erweiterung in anderen Bereichen keinen großen Aufwand dar.

Abkürzungsverzeichnis

AT ..... *Allgemeiner Teil*

BLOB ..... *Binary Large Object*

BSI ..... *Bundesamt für Sicherheit in der Informationstechnik*

COREP ..... *Common Reporting Framework*

CSP ..... *Cloud Service Provider*

CSV ..... *Comma-separated values*

DOM ..... *Document Object Model*

EBA ..... *European Banking Authorit*

IaaS ..... *Infrastructure as a Service*

ITS ..... *Implementing Technical Standards*

JDBC ..... *Java Database Connectivity*

JSF ..... *JavaServer Faces*

KWG ..... *Kreditwesengesetz*

MaRisk ..... *Mindestanforderungen an das Risikomanagement*

NIST ..... *National Institute of Standards and Technology*

PaaS ..... *Platform as a Service*

RaaS ..... *Reporting as a Service*

RegEx ..... *Regular Expression*

SaaS ..... *Software as a Service*

SLA ..... *Service-Level-Agreement*

SOA ..... *Service-oriented Architecture*

SQL ..... *Structured Query Language*

VM ..... *Virtuelle Maschine*

XBRL ..... *eXtensible Business Reporting Language*

XML ..... *Extensible Markup Language*

XLS/XLSX ..... *Excel Spreadsheet*

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literaturverzeichnis

2010 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT 2010). Seoul.

2010 proceedings, IEEE INFOCOM. San Diego, California, USA : 15-19 March 2010 (2010). [Piscataway, N.J.]: IEEE Xplore.

BaFin (2012): Mindestanforderungen an das Risikomanagement. An alle Kreditinstitute und Finanzdienstleistungsinstitute in der Bundesrepublik Deutschland 2012, zuletzt geprüft am 06.10.2015.

Barton, Thomas (2014): E-Business mit Cloud Computing. DOI: 10.1007/978-3-8348-2426-4.

Baun, Christian; Kunze, Marcel; Nimis, Jens; Tai, Stefan (2011): Cloud Computing. Berlin, Heidelberg: Springer Berlin Heidelberg.

BearingPoint Software Solutions GmbH (2015): ABACUS/DaVinci. Online verfügbar unter http://www.bessgmbh.com/de/produkte/abacusdavinci/.

Böttger, Markus (2012): Cloud Computing richtig gemacht. Ein Vorgehensmodell zur Auswahl von SaaS-Anwendungen : am Beispiel eines hybriden Cloud-Ansatzes für Vertriebssoftware in KMU. Hamburg: Diplomica Verlag, zuletzt geprüft am 28.09.2015.

BSM GmbH (2015): BAIS - Die Meldewesensoftware der zufriedenen Kunden. Online verfügbar unter https://www.bsmgmbh.de/bais/.

Bundesamt für Sicherheit in der Informationstechnik (2014a): Goldene Regeln Baustein B 1.17 Cloud-Nutzung.

Bundesamt für Sicherheit in der Informationstechnik (2014b): Goldene Regeln Baustein B 5.23 Cloud Management.

Bundesamt für Sicherheit in der Informationstechnik (2014c): IT-Grundschutz-Kataloge 14. Ergänzungslieferung.

Bundesbank (2016a): Großkredit-Plausibilitätsprüfungen. Hg. v. Bundesbank. Frankfurt am Main. Online verfügbar unter https://www.bundesbank.de/Redaktion/DE/Downloads/Service/Meldewesen/Bankenaufsicht/PDF/plausibilitaetsliste_grosskredit_1.pdf?__blob=publicationFile.

Bundesbank (2016b): Obergrenzen für Großkredite (LE Limits). Online verfügbar unter https://www.bundesbank.de/Redaktion/DE/Downloads/Service/Meldewesen/Bankenaufsicht/PDF/le_limits.pdf?__blob=publicationFile.

Bundesbank, Deutsche (2009): Merkblatt für die Abgabe der Groß- und Millionenkreditanzeigen nach §§ 13 bis 13b und 14 KWG, zuletzt geprüft am 02.10.2015.

Bundesbank, Deutsche (2014): Verordnung zur Ergänzung der Großkreditvorschriften nach der Verordnung (EU) Nr. 575/2013 des Europäischen Parlaments und des Rates vom 26. Juni 2013 über Aufsichtsanforderungen an Kreditinstitute und Wertpapierfirmen und zur Änderung der Verordnung (EU) Nr. 646/2012 und zur Ergänzung der Millionenkreditvorschriften nach dem Kreditwesengesetz (Großkredit- und Millionenkreditverordnung – GroMiKV), zuletzt geprüft am 07.10.2015.

Bundesbank, Deutsche (2015a): Gesetz über das Kreditwesen - KWG. Online verfügbar unter https://www.bundesbank.de/Redaktion/DE/Downloads/Aufgaben/Bankenaufsicht/Gesetze_Verordnungen_Richtlinien/gesetz_ueber_das_kreditwesen_kwg.pdf?__blob=publicationFile, zuletzt geprüft am 02.10.2015.

Bundesbank, Deutsche (2015b): Extranet V 2.13.

Cloud Foundry (2016): Cloud Foundry Documentation. Online verfügbar unter http://docs.cloudfoundry.org/.

CoreFiling (2013): SEAHORSE XBRL REPORTING – NOW.

Flickinger, Norbert (2013): XBRL in der betrieblichen Praxis. Der Standard für Unternehmensreporting und E-Bilanz. 2., neu bearb. Aufl. Berlin: ESV - Erich Schmidt Verlag GmbH & Co. KG, zuletzt geprüft am 29.09.2015.

Günther, Frank (2014): Handbuch Bankaufsichtliches Meldewesen. Vorgaben - Datenanforderungen - Umsetzungshinweise. 2. Aufl. Heidelberg: Finanz-Colloquium Heidelberg (Kredit, Immobilien).

Hoffman, Charles; Watson, Liv Apneseth (2010): XBRL for dummies. Hoboken, NJ: Wiley Pub (--For dummies). Online verfügbar unter http://site.ebrary.com/lib/academiccompletetitles/home.action.

HPE Helion Stackato (2016): HPE Helion Stackato Documentation. Online verfügbar unter docs.stackato.com.

Institute of Electrical and Electronics Engineers; International Conference on Artificial Intelligence, Management Science and Electronic Commerce; AIMSEC (2011): 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011. Zhengzhou, China, 8 - 10 August 2011. Piscataway, NJ: IEEE. Online verfügbar unter http://ieeexplore.ieee.org/servlet/opac?punumber=5992814.

Kavis, Michael (2014): Architecting the cloud. Design decisions for cloud computing service models (SaaS, PaaS, and IaaS). Hoboken: Wiley (Wiley CIO). Online verfügbar unter http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=817374.

Lan, Hua (Hg.) (2014): International Conference on Management Science & Engineering (ICMSE), 2014. 17 - 19 Aug. 2014, Helsinki, Finland ; 21st annual conference proceedings. Institute of Electrical and Electronics Engineers; Aalto-Yliopisto; International Conference on Management Science and Engineering; ICMSE. Piscataway, NJ: IEEE. Online verfügbar unter http://ieeexplore.ieee.org/servlet/opac?punumber=6917165.

Niemann, Fabian; Paul, Jörg-Alexander; Bieber, Nicolai (Hg.) (2014): Rechtsfragen des Cloud Computing. Herausforderungen für die unternehmerische Praxis. Berlin: De Gruyter (De Gruyter Praxishandbuch).

NIST Computer Security Division (CSD): NIST SP 800-145, The NIST Definition of Cloud Computing, zuletzt geprüft am 30.09.2015.

Red Hat (2016): OpenShift Origin Documentation Site. Online verfügbar unter https://docs.openshift.org/origin-m4/index.html.

Schicker, Edwin (2014): Datenbanken und SQL. Wiesbaden: Springer Fachmedien Wiesbaden.

Sosinsky, Barrie (2011): Cloud computing bible. [explore the cloud with this complete guide ; understand all platforms and technologies ; use Google, Amazon, or Microsoft web services]. Indianapolis, Ind.: Wiley (Bible, v.757). Online verfügbar unter http://www.eblib.com/patron/FullRecord.aspx?p=706883.

Vossen, Gottfried; Haselmann, Till; Hoeren, Thomas (2012): Cloud-Computing für Unternehmen. Technische, wirtschaftliche, rechtliche und organisatorische Aspekte. 1. Aufl. Heidelberg: dpunkt-Verl. (Safari Tech Books Online). Online verfügbar unter http://proquest.safaribooksonline.com/9781492000945.